Rechte, Benutzerrollen und Inhaltsversionierung für verteilte Multimedia-Autorensysteme

Konzeption und Realisierung

vom Fachbereich 20
der Technischen Universität Darmstadt
genehmigte Dissertation
zur Erlangung des Grades eines
Doktor-Ingenieur (Dr.-Ing.)

von

Diplom-Informatiker Michael Liepert

geboren am 12.3.1966 in Bobingen

Darmstadt 2001
Hochschulkennziffer D17

Vorsitzender:	Prof. Dr.-Ing. Dr. h.c. Dr. E.h. José Encarnção
Erstreferent:	Prof. Dr.-Ing. Ralf Steinmetz
Korreferent:	Prof. Dr. Erich J. Neuhold

Tag der Einreichung:	15.Oktober 2001
Tag der Disputation:	14. Dezember 2001

Eidesstattliche Erklärung

Ich versichere hiermit an Eides statt, dass ich die vorliegende Arbeit allein und nur unter Verwendung der angegebenen Literatur verfasst habe.

Darmstadt, den 14. Oktober 2001

_____

Dipl.-Inform. Michael Liepert

Inhaltsverzeichnis

Inhaltsverzeichnis iii

Abbildungsverzeichnis vi

Tabellenverzeichnis viii

Kapitel 1 - Einführung 1

1.1 Multimediale Anwendungen mit vielen Autoren 1
1.2 Ziele der Arbeit 3
1.3 Vorgehen 3
1.4 Struktur der Arbeit 3

Kapitel 2 - Problemanalyse 5

2.1 Beispielapplikationen 5
2.1.1 Kiosksystem 5
2.1.2 News-On-Demand-System 6
2.1.3 Vorlesungsarchiv 7
2.2 Die Arbeitsziele 8
2.3 Abgrenzung der Arbeit 10

Kapitel 3 - Analyse des Umfelds 11

3.1 Verteilte Multimedia-Applikationen 11
3.1.1 Prototypen 12
3.1.2 MPEG-4: Mehr als ein Medienformat 12
3.1.3 WWW: Mehr als Hypertext 15
3.1.4 CORBA: Mehr als Middleware 26
3.2 Rechtemanagement 27
3.2.1 WebDAV 27
3.2.2 UNIX 31
3.2.3 AFS 32
3.2.4 Windows 2000 33
3.3 Versionierung und Synchronisation der Arbeit vieler Autoren 35
3.3.1 Grundlagen des Configuration Management 36
3.3.2 Ausgewählte Verfahren zur Versionskontrolle in DBMS 46
3.3.3 CSCW und Workflow 54
3.4 Schlussfolgerung 54

Kapitel 4 - Architektur und logische Datenorganisation 56

4.1 Architektur 57
4.1.1 Anforderungen 57
4.1.2 Realisierung 60
4.2 Anforderungen an die Datenstruktur 61

4.2.1 Suchen und Sammeln von Inhalten ... 61
4.2.2 Import und Export von Inhalten ... 61
4.2.3 Bearbeitung von Inhalten ... 62
4.2.4 Strukturierung von Inhalten ... 62
4.2.5 Trennung von Inhalten und Layout ... 63
4.2.6 Rechtemanagement ... 63
4.2.7 Unterstützende Funktionalitäten ... 63
4.2.8 Quality of Service ... 64
4.2.9 Alternative Inhalte ... 64
4.2.10 Signalisierungsweg für Systemdienste ... 64
4.2.11 Integration mit physikalischen Gegebenheiten ... 64
4.2.12 Zusammenfassung ... 65
4.3 Konzeption der Datenstruktur ... 65
4.3.1 Zentrale Archivierung ... 65
4.3.2 Trennung von Inhalt, Layout und Struktur ... 66
4.3.3 Der Ressourcenbaum ... 66
4.3.4 Die Layoutliste ... 67
4.3.5 Die Präsentationsliste ... 67
4.3.6 Metadaten zur Dokumentation ... 68
4.3.7 Dynamische Erzeugung von Präsentationen ... 68
4.3.8 Alternative Medien ... 69
4.3.9 Multimediale Referenzen ... 70
4.3.10 Konvertierungsmodule ... 70
4.3.11 Plattform ... 71
4.3.12 Liste der zentralen Entscheidungen ... 71
4.4 Realisierung der Datenstruktur ... 72
4.4.1 Technisches Konzept ... 72
4.4.2 Schnittstellen des Datenorganisation ... 73
4.4.3 Datentyp ... 76
4.5 Zusammenfassung ... 79

Kapitel 5 - Rechtemanagement ... 81

5.1 Anwendungsbeispiele ... 82
5.1.1 Urheberrechte ... 82
5.2 Zentrale Entscheidungen ... 83
5.2.1 Ziele ... 83
5.2.2 Konsequenzen für das zu wählende Rechtemanagement ... 84
5.3 Vorgeschlagenes Rechtemanagementsystem ... 87
5.3.1 Konzepte ... 87
5.3.2 Systemvorgaben ... 93
5.3.3 Benachrichtigungen ... 94
5.3.4 Spezifizierung des Rechtekonzeptes ... 94
5.3.5 Realisierung urheberrechtlicher Aspekte ... 95

Kapitel 6 - Nutzer- und Autoren-Rollen ... 97

6.1 Motivation ... 97
6.1.1 Beispiel I: Kiosksystem ... 97
6.1.2 Beispiel II: News-On-Demand-System ... 98

6.1.3 Beispiel III: Vorlesungsarchiv 99
6.2 Anforderungen an ein Rollenkonzept 100
6.3 Definition des Rollenkonzepts 101
6.3.1 Rollen 102
6.3.2 Systemvorgaben 104
6.3.3 Semantik 106
6.4 Bewertung des Rollenkonzeptes 107
6.4.1 Machbarkeit 108
6.4.2 Vollständigkeit 108
6.4.3 Komplexität 108

Kapitel 7 - Versionierung 110

7.1 Zielanalyse 110
7.1.1 Nutzungsszenarien 111
7.1.2 Fazit: Auswertung der Szenarien 118
7.2 CM- und DBMS-Versionierungsmodelle 120
7.2.1 Versionierung in CM-Systemen 121
7.2.2 Mechanismen in Datenbank-Systemen 121
7.2.3 Relevanz der Mechanismen 125
7.3 Spezifikation des Ziel-Versionierungssystems 125
7.3.1 Versionsbaum für Ressourcen, Präsentationen und Layouts 127
7.3.2 Änderung der Datenstruktur von Referenzen 128
7.3.3 Erweiterung des Nutzerprofils 129
7.3.4 Spezifizierung der Versionsschnittstelle 130
7.4 Prototypische Implementierung 139
7.4.1 Durchgeführte XML-Transformationen und ihre Spezifikation 141
7.5 Zusammenfassung 141

Kapitel 8 - Zusammenfassung und Ausblick 145

Anhang A - Referenzen 147

Anhang B - Abkürzungen 156

Anhang C - Zur Datenstruktur 159

Anhang D - Zum Rollenkonzept 162

D.1 Ableitung der Rollen-Zugriffsrechte 162
D.2 Darstellung der Metadaten 168

Anhang E - Zur Versionierung 170

E.1 Prototypische Anfragesprache 170
E.2 Typ-Definition der Versionierungsdaten 171

Abbildungsverzeichnis

Kapitel 1 - Einführung 1

Kapitel 2 - Problemanalyse 5

Abbildung 1: Arbeitsschritte und -rollen des NoD-Systems HyNoDe 6

Kapitel 3 - Analyse des Umfelds 11

Abbildung 2: eine MPEG-4 Session mit zwei MPEG-4-Terminals 13
Abbildung 3: Beispiel für die Zusammensetzung einer MPEG-4-Szene aus Audio-Visuellen-Objekten (AVO) 14
Abbildung 4: Versionierungsrepräsentation in einem Versionengraph 39
Abbildung 5: Datenmodell eines CAD-DBMS nach [25] 48
Abbildung 6: Currency Mechanismus in [25] 49
Abbildung 7: Change Propagation nach [25] 51
Abbildung 8: Entwicklungsstufen eines Dokumentes 53

Kapitel 4 - Architektur und logische Datenorganisation 56

Abbildung 9: Kapselungen der Sicht auf die Datenhaltung 60
Abbildung 10: Architektur und Beispielmodule 60
Abbildung 11: Alternative Medien und Sequenzen im Ressourcenbaum 69
Abbildung 12: Technisches Konzept der Datenorganisation im Überblick 72
Abbildung 13: Schnittstellen des Datenformats 74
Abbildung 14: Oberste Ebene der Datenorganisation 77
Abbildung 15: Inhaltsbereich des Datenorganisation 78
Abbildung 16: Die Struktur des Ressourcenbaums 78
Abbildung 17: Die Struktur der Layoutliste 79
Abbildung 18: Die Struktur der Präsentationsliste 79

Kapitel 5 - Rechtemanagement 81

Abbildung 19: Rekursive Organisationsstruktur 89

Kapitel 6 - Nutzer- und Autoren-Rollen 97

Abbildung 20: Meta-Schema einer relationalen Datenbank (Entity-Relationship) 103
Abbildung 21: Meta-Schema: erweitert um Rollen-ACLs (rACLs) 103
Abbildung 22: Rollen-ACLs für "Autor" und konkrete Inhalte 104

Kapitel 7 - Versionierung 110

Abbildung 23: Datenstrukturerweiterungen in Ressourcen- Präsentations- und Layoutknoten 128
Abbildung 24: Nutzerprofil für Historie der benutzten Präsentationen 130
Abbildung 25: Notifikationsdatenstruktur im Nutzerprofil 131

Abbildung 26: Gewählter Update Mechanismus 133
Abbildung 27: Update der Knoten über die Versionsschnittstelle 134
Abbildung 28: Currency 135
Abbildung 29: Aktualisierung der Referenzen bei Erzeugung von Seitenästen 137
Abbildung 30: Zielarchitektur des Versionskontrollsystems 140

Kapitel 8 - Zusammenfassung und Ausblick 145

Tabellenverzeichnis

Kapitel 1 - Einführung 1

Kapitel 2 - Problemanalyse 5

Kapitel 3 - Analyse des Umfelds 11

Tabelle 1: Bewertung von MPEG-4 als Grundlage verteilter MM-Autorensysteme 15
Tabelle 2: Bewertung des WWW als Grundlage verteilten Multimedia-Authorings 25
Tabelle 3: Vorhandene Middleware als Grundlage verteilten MM-Authorings 27

Kapitel 4 - Architektur und logische Datenorganisation 56

Tabelle 4: Die im weiteren addressierten Probleme aus dem Themenumfeld 56

Kapitel 5 - Rechtemanagement 81

Kapitel 6 - Nutzer- und Autoren-Rollen 97

Kapitel 7 - Versionierung 110

Tabelle 5: Benachrichtigungen bei entfernten Updates 115
Tabelle 6: Nutzungszenarien und abgeleitete benötigte Funktionalität 118
Tabelle 7: Benötigte Funktionalität 119
Tabelle 8: Zu erweiternde Bereiche der Datenstruktur 120
Tabelle 9: Drei CM-Tools im Überblick 122
Tabelle 10: Relevante CM-Mechanismen 123
Tabelle 11: Relevante Mechanismen von DBMS 124
Tabelle 12: Relevante Mechanismen: CM und DBMS kumuliert 126

Kapitel 8 - Zusammenfassung und Ausblick 145

Never give up
— lipi

Kapitel 1 - Einführung

1.1 Multimediale Anwendungen mit vielen Autoren

Rechner und Netze sind seit einigen Jahren besser in der Lage, multimediale Daten zu speichern, zu transportieren und zu präsentieren [49]. Entsprechend werden digitale Medien zunehmend auch im kommerziellen Umfeld genutzt. Gleichzeitig steigt der Anspruch an Umfang und Qualität des medialen Inhalte s mit der Zahl der vorhandenen Systeme.

Beispiele für solche kommerziellen Anwendungen sind umfangreiche kommerzielle Web-Auftritte, News-On-Demand-Systeme (*NoD*-Systeme) oder Kiosksysteme (*POI*: Point of Information, *POS*: Point of Sale).

Bei multimedialen Projekten lässt sich der Realisierungsaufwand in die Inhaltserstellung und in die Realisierung des Systems selbst aufteilen. Durch hohe Anforderungen an den Inhalt, an seine Aktualität und Konsistenz und an die Qualität der Präsentation kann der Aufwand dafür den für die Systemerstellung weit überwiegen. Dies begünstigt dann das Auftreten eines Engpasses bei der Person, die für die Integration der Resultate aller Autoren (das sind eventuell viele Agenturen, Journalisten, Redakteure und Designer) verantwortlich ist. Noch ist es schwierig, Teile dieser Integrationsaufgabe an andere delegieren: es fehlen viele Mechanismen, um diesen Engpass zu entschärfen und eine klare, zuverlässige und nachvollziehbare Delegation der Inhaltserstellung zu ermöglichen.

Diese Arbeit bringt einen Beitrag zur maschinellen Unterstützung der Integration der multimedialen Erzeugnisse vieler Autoren, um diese skalierbarer zu machen. Dazu zeigt diese Arbeit, dass bei multimedialen Systemen mit vielen Autoren andere Eigenschaften wichtig sind als bei nicht-multimedialen Systemen mit vielen Autoren. Desweiteren wird aufgezeigt, dass die vorhandenen Techniken zur Erstellung großer multimedialer Anwendungen für viele Benutzer noch Lücken bei der Unterstützung vieler Autoren offen lassen.

Auf Grundlage einer möglichst einfachen Architektur und einer flexiblen Datenstruktur adressiert die Arbeit dann die drei wichtigsten offenen Probleme auf Applikationsebene: eine adäquate Rechteorganisation und -semantik für die Koordination der Autoren, ein neues Konzept zur Verwirklichung von Autoren-Rollen und eine an die Anforderungen multimedialer Inhalte angepasste Versionierung der Inhalte.

Rechtesysteme von Dateihierarchien haben eine einfache Struktur, meistens besteht diese nur aus Verzeichnissen, Dateien und deren Metadaten. Sie erlauben zwar die Definition von Benut-

zern und Gruppen, die Vergabe der Rechte ist jedoch zu grobkörnig. So genannte Directory Services erlauben durch eine genügend fein granulierte Rechtevergabe, Rechte adäquat für die reiche Struktur von Multimedia-Inhalten zu realisieren, ohne allerdings bisher für diese Struktur eine entsprechende Rechte-Semantik zu definieren. Eine solche wird in dieser Arbeit entwickelt. Ein wichtiger Punkt neben der reinen Anpassung ist dabei die Erweiterung um spezifische Eigenheiten multimedialer Inhalte. Ein Beispiel für solche speziellen Eigenschaften ist die Notwendigkeit, oft ganz spezielle Begrenzungen der Nutzung und Nutzungsarten im System zu beschreiben und zu kontrollieren. Hierzu kann die Anzahl der Aufrufe eines bestimmten Textes durch einen Benutzer zählen oder die zeitlich begrenzte Lizenzierung eines Videos an eine Nutzergruppe.

Multimediales Arbeiten bringt häufig verschiedene **Arbeitsrollen** hervor: etwa den Corporate Designer und den Designer der normalen Inhalte, den Editor und den Regisseur. Die Charakteristik solcher Rollen bestimmt die Zusammenarbeit innerhalb von Teams. In großen, verteilten Multimediasystemen können heterogene, verschieden organisierte Teams auftreten. Diese Arbeit entwickelt und untersucht eine Möglichkeit, solche Rollen unabhängig von Zugriffsrechten in Teams dezentral, dennoch explizit und verwaltbar, zu realisieren. Bestehende Systeme organisieren Rollen nicht orthogonal zur Rechteverwaltung, die meisten Systeme kennen keine explizit für jeden Benutzer wiederverwendbaren Rollen.

Als Beispielanwendung dient in dieser Arbeit ein verteiltes Archiv für multimediale Vorlesungsinhalte. Die wichtigsten Nutzer eines solchen Systems sind Lehrende mit meist kleinen Autorenteams, die dezentral ihre spezifischen Arbeitsvorgänge und -aufteilungen festlegen. Diese Strukturen muss ein solches Archiv unterstützen, will es effizient einsetzbar sein. Wollen die Professoren aber innerhalb dieses Systems mit anderen Lehrstühlen kooperieren, müssen die verschiedenen Organisationen im System abbildbar und vereinbar sein. Konkre heißt das, dass ein Mitarbeiter eines Professors Materialien eines anderen Professors nutzen können sollte. Das System sollte dabei automatisch die Vorgaben des Chefs des Mitarbeiters zur Erstellung von Präsentationen und die Vorgaben des anderen Professors zur Arbeit innerhalb seines Bereichs unterstützen.

Versionsmanagement ist ein wichtiger Begriff bei der Archivierung von Inhalten, etwa in Datenbanken oder Systemen zur Koordinierung in Programmiererteams. Ähnlich wie beim oben genannten Rechtemanagement unterscheidet sich die multimediale Anwendung aber von den bestehenden Systemen. Autoren bearbeiten hier viele Arten von Inhalten (Text, Audio, Video, Hypertext und deren Metadaten) und ihre reich strukturierten Inhaltsbeziehungen, etwa Hypertext-Referenzen, inhaltliche Äquivalenz und Enthaltensein. Diese Strukturen können sehr reich sein, gleichzeitig sind die geforderten Eigenschaften an die Versionierung formal weniger streng als etwa bei archivierten Programmtexten. Das im weiteren an multimediale Anwendungen angepasste Versionsmanagement zeichnet sich dementsprechend durch ein angepasstes Notifizierungskonzept und ein vereinfachtes Change-Management aus, um die Arbeit an multimedialen Inhalten effizient aber nachvollziehbar zu gestalten.

Ändert ein Mitarbeiter eines Professors eine Tabelle in seiner Vorlesung, die in der Vorlesung eines anderen Professors genutzt wird, sind beide Professoren betroffen. Dennoch wollen beide so wenig wie möglich davon behelligt werden. Die Mechanismen vor allem von Daten-

banken und Konfigurationsmanagement-Systemen sind mächtiger als die hier geforderten Lösungen. Deswegen werden in dieser Arbeit diese Mechanismen in einigen Aspekten entsprechend (teilweise konfigurierbar) spezialisiert.

1.2 Ziele der Arbeit

Ziel der Arbeit ist es aufzuzeigen, dass heutige Techniken multimediale Systeme mit vielen Autoren unzureichend unterstützen. Desweiteren werden Mechanismen entwickelt oder angepasst, um mehr Unterstützung, speziell bei den Themen adäquates Rechtemanagement, Benutzerrollen, Versionierung, zu realisieren.

Durch den Vorschlag einer Architektur und einer Datenorganisation und die anschließdende Erarbeitung und Integration der Lösungen für aufgezeigte Probleme soll dargelegt werden, dass und wie eine explizite und angepasste Datenorganisation genutzt werden kann, notwendige Mechanismen für verteilte multimediale Systeme mit vielen Autoren zu realisieren.

1.3 Vorgehen

Diese Arbeit betrachtet Systeme, die multimediale Fähigkeiten mit der Fähigkeit kombinieren, viele Autoren zu unterstützen. Die beiden kombinierten Gebiete sind umfangreich, sie sind seit langem in der praktischen Anwendung und beinhalten so inzwischen eine Vielfalt von Aspekten, die sich aus dem praktischen Einsatz der vorhandenen technischen Lösungen ergeben.

Diese Arbeit geht nicht rein analytisch vor, sondern wählt den folgenden pragmatischen und konstruktiven Ansatz: drei reelle multimediale Projekte mit vielen Inhaltsautoren werden vorgestellt, das umfassendste wird als Zielapplikation gewählt. Dann werden Lücken der vorhandenen Techniken durch Diskussion im Hinblick auf diese Zielapplikation identifiziert.

Um diese Lücken zu schließen, wird eine möglichst einfache Architektur als gemeinsame Basis zur Erstellung und Integration eigener, angepasster Lösungen verwendet. Diese Lösungen wurden teilweise im Rahmen des Projekts Medianode implementiert und demonstriert, teilweise wurden sie zur grundsätzlichen Bestätigung der Machbarkeit der Lösungen als Prototypen auf Teilmengen desselben Projektes realisiert.

1.4 Struktur der Arbeit

Nach dieser Einleitung wird das Problemfeld der Arbeit zunächst in Kapitel 2 genauer betrachtet und die anzustrebenden Ziele konkretisiert.

In Kapitel 3 folgt eine umfangreiche Untersuchung des thematischen Umfeldes, also einerseits von Forschungsarbeiten und vorhandenen Techniken für verteilte Multimedia-Anwendungen und andererseits für Anwendungen mit vielen Autoren. Das sind zum einen Middlewareansätze wie CORBA, Frameworks für Multimedia-Anwendungen wie die MPEG-Standards, aber auch Themen aus den Bereichen Konfigurations- und Versionsmanagement, Datenbanken und Workflow.

In den weiteren Kapiteln werden dann konstruktiv Lösungen präsentiert, wobei sich der Fokus sich aus der Analyse in den ersten Kapiteln der Arbeit ergibt: Konzepte, die multimedia-

len Systemen mit vielen Autoren adäquat Benutzerrechte, Benutzerrollen und eine passende Versionierung der Inhalte bieten.

In Kapitel 4 werden zunächst eine Architektur und eine Struktur der Anwendungsdaten für ein multimediales Beispielsystem mit vielen Autoren entwickelt. Eine wichtiges Entwurfsziel hierbei ist, dass das Resultat tragfähig für eine Erweiterung um die in in der ersten Hälfte der Arbeit identifizierten zu adressierenden Konzepte ist.

Ausgehend von den in Kapitel 3 vorgestellten Arbeiten werden die Themen Rechtemanagement, Benutzerrollen und Versionierung in dieser Reihenfolge jeweils in Kapitel 5, 6 und 7 bearbeitet. In jedem dieser Kapitel wird ein Satz von Zielen aus den Anforderungen festgelegt und dazu eine Lösung auf Basis des gewählten Systems konstruiert. Während die Konzeption für Rechte und Versionen in den Kapiteln 5 und 7 weitgehend durch Anpassung und Kombination vorhandener Ansätze gewonnen wird, wird das Rollenkonzept in Kapitel 6 dagegen im Ansatz verschieden von bisher verwendeten Systemen entwickelt.

Abschließende Betrachtungen fassen die Ergebnisse in Kapitel 8 zusammen und geben einen Ausblick auf weitere mögliche Arbeiten.

Kapitel 2 - Problemanalyse

In diesem Kapitel Zunächst soll das Problemumfeld und die Heransgehensweise näher bestimmt werden. Dazu werden zuerst drei existierende Projekte vorgestellt, die das adressierte Problemumfeld umreißen: die Klasse der verteilten multimedialen Applikationen, die die Arbeit vieler verschiedener, für den Applikationsinhalt verantwortlicher Personen integrieren müssen.

Zunächst werden nun die Aufgaben, die in diesem Umfeld auftreten, näher betrachtet.

2.1 Beispielapplikationen

Die drei folgenden Beispiele beschreiben reelle Projekte, sie sind aus verschiedenen Bereichen, haben aber die Eigenschaft, dass die Applikationsinhalte von vielen Autoren erzeugt und gepflegt werden, gemeinsam.

Die folgenden Beispiele, besonders das dritte aus Abschnitt 2.1.3, sollen auch im weiteren Lauf der Arbeit dazu dienen, Probleme und Lösungen im einzelnen konkret zu diskutieren.

2.1.1 Kiosksystem

In [51] wird ein interaktives, multimediales Kiosksystem vorgestellt, welches den Input von hunderten bis tausenden Autoren in kommerziellen Installationen verbindet. Beim Einsatz dieses Systems zum Beispiel bei Banken ist die inhaltliche und formale Integrität und Konsistenz der Anwendung oberstes Ziel und die Benutzbarkeit durch die verschieden ausgebildeten Autoren eine unabdingbare Rahmenbedingung.

[51] kommt zu dem Schluss, dass bei einer Kioskanwendung, die zentrales Design, zentrale Inhalte eines Konzerns und Inhalte Tausender von Filialen verbinden muss, für die Autoren verschiedene Editierschnittstellen notwendig sind und beschreibt einige Autorenrollen, die von einer solchen Anwendung unterstützt werden müssen.

Die beteiligten Autoren gliedern sich grob in drei Gruppen:

- Applikationsdesigner: es ist ein verantwortlicher Applikationsdesigner, der die Grundstruktur der Applikation vorgibt. Das Applikationsdesign wird nur in Einzelfällen geändert. Das Applikationsdesign erzeugt initial Form, Struktur und Inhalt der Applikation.
- Autoren des Corporate Design: wenige Autoren erstellen und Pflegen das Corporate Design. Diese Tätigkeit findet kontinuierlich statt und besteht hauptsächlich aus der Pflege formaler Elemente der Applikation wie Symbolen und formalen Attributen (Schriftgröße, Farben, etc.).
- Inhaltsautoren: sehr viele Inhaltsautoren ändern, erstellen und löschen Inhalte.

Weiterhin werden die Rollen des Benutzers oder Konsumenten der Anwendung quasi als “Null-Autoren” ebenso in dieses Modell aufgenommen wie die Rolle eines Programmierers,

der externe Anwendungen integriert. Die innerhalb des Kiosksystems anfallende Tätigkeit des letzteren erfolgt durch das Setzen von Attributen des multimedialen Inhaltes durch die ersten beiden Autorenrollen.

Die Autoren stellen fest, dass für ihre Anwendung im kommerziellen Umfeld die Gesamtapplikation sehr hohen formalen und inhaltlichen Konsistenzkriterien genügen muss, und dass die Arbeiten besonders der zweiten und der dritten Gruppe der Autoren regelmäßig jeweils sehr verschiedene Operationen vom System verlangen bzw. ausschließen müssen. Insbesondere müssen den Corporate Designern praktisch alle Möglichkeiten zum freien formalen Ändern gegeben werden, während die Möglichkeit zum freien formalen Ändern das entsprechende Autorenprogramm praktisch unbrauchbar für die vielen, wenig geschulten Inhaltsautoren machen würde.

2.1.2 News-On-Demand-System

Im Rahmen des News-On-Demand-Systems (*NoD*-Systems) *HyNoDe* wurde der multimediale Input vieler Autoren (Fotografen, Journalisten) durch Redakteure verschiedener Inhaltsanbieter - Presseagenturen, Zeitungsverlage - über einen Dienstanbieter koordiniert an die Endkunden weitergereicht. Als kommerzieller Nachrichtenanbieter waren neben den Kriterien der Inhalts- und Formkonsistenz jeweils für jeden der Inhaltsanbieter die Aktualität der Nachrichten und die Wiederverwendbarkeit von Monomedien-Bauteinen entscheidende Kriterien.

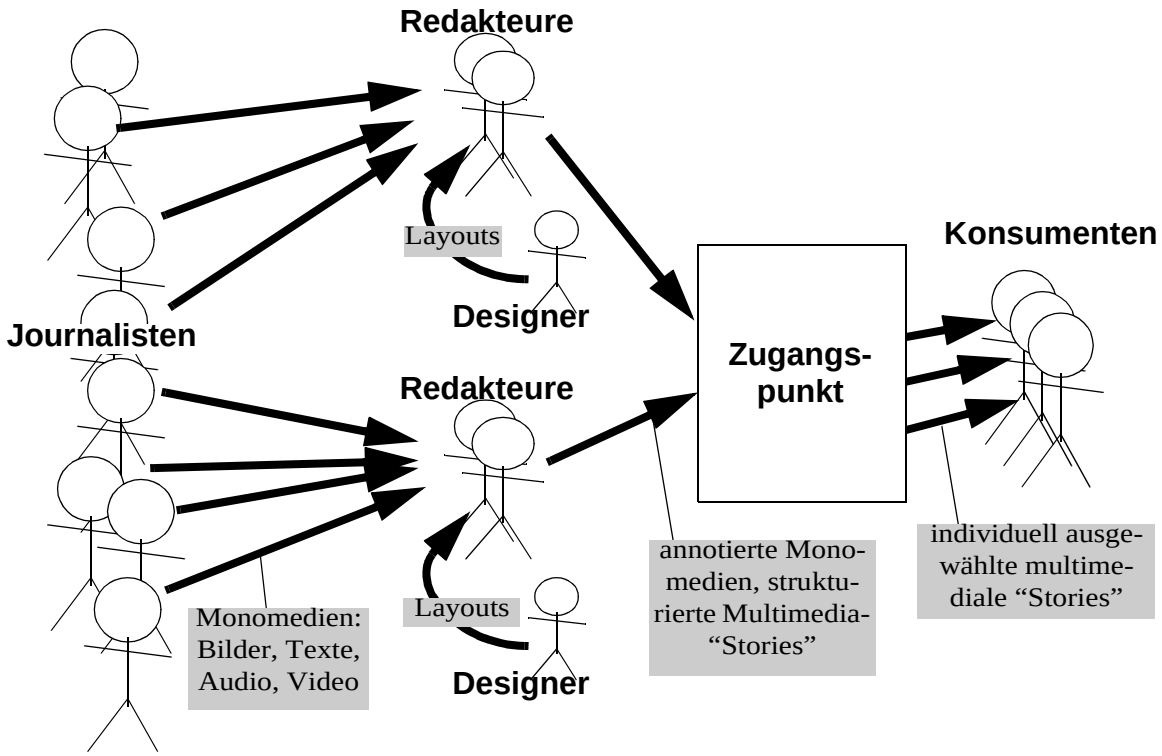

Abbildung 1: Arbeitsschritte und -rollen des NoD-Systems HyNoDe

Abbildung 1 zeigt die Arbeitsschritte, die eine Analyse des Arbeitsumfeldes der Journalisten und Redakteure ergab, und welche durch entsprechende Werkzeuge zu unterstützende Autorenrollen:

Im einzelnen sind das:

- Journalisten
  als Ersteller von (monomedialen) Inhalten
- Designer
  als Autoren von (multimedialen) Layouts
- Redakteure
  als Struktur-Editoren, die Zusammenhänge zwischen Monomedien und deren Attribute, insbesondere Zuordnung zum entsprechenden Layout und Einordnung in die Taxonomie des jeweiligen Inhaltsanbieters schaffen

Ein System wie HyNoDe muss es leisten, Gruppen von Autoren in voneinander unabhängigen Einheiten (etwa Agenturen, Redaktionen) ebenso wie den Austausch und die Wiederverwendung zwischen diesen Einheiten zu koordinieren. Für ein verteiltes Kiosksystem wie in Abschnitt 2.1.1 kann eine strenge, uniforme, hierarchische Organisation der Autoren innerhalb einer zusammenhängenden, kooperativen Umgebung angenommen werden. In oberen Hierarchiestufen wird Design erstellt, welches mit globalem bzw. zunehmend lokalen und temporären Inhalten auf tieferen Ebenen kombiniert wird.

Die in einem System wie HyNoDe existierenden voneinander unabhängigen Inhaltsanbieter und Agenturen machen Verwaltung, Authentisierung und organisatorische Zuordnung (etwa zu Firmen) der beteiligten Autoren wichtiger. Die Interaktionen zwischen den Systeminstanzen bei den beteiligten Organisationen jedoch benötigen kein Wissen über die Autoren, so dass Organisationsstrukturen nur lokal innerhalb einer einzelnen Organisation verwendet werden müssen. (Die systemweite Verwaltung und Kommunikation des Feldes *Autor* für Mono- und Multimedien ist keine Ausnahme hiervon, da es nicht zum Systembetrieb genutzt, sondern nur als informatives Datenfeld für den Endkunden zur Verfügung gestellt wird.)

2.1.3 Vorlesungsarchiv

Medianode ist ein Projekt des Hessischen Telemedia Technologie Kompetenz-Centers *httc*, das von Mitte 1998 bis Herbst 2001 am Lehrstuhl KOM der TU Darmstadt bearbeitet wurde [34][77][77]. Das Projekt zielt auf eine Infrastruktur, die Erstellung, Nutzung und Austausch von multimedialem Lehrmaterial unterstützt.

Das große Problem bei der Nutzung multimedialen Lehrmaterials besteht in dem Verhältnis des Erstellungsaufwands bzw. der Erstellungsdauer zur Qualität und Verwendungsdauer des Materials. Multimediales Material wird trotz vorhandener Möglichkeiten nur sehr selten direkt in Vorlesungen und Vorträge eingebettet. Medianode stellt keine zusätzlichen Werkzeuge für die Erstellung des Materials zur Verfügung, sondern unterstützt die Arbeitsteilung bei der Erstellung mit vorhandenen Mitteln und die Wiederverwendung existierenden Materials. Da eine Zusammenarbeit zwischen den Lehrenden durchaus bereits existiert, wird erwartet, dass die entstehende Infrastruktur für die einfache Zusammenarbeit über Universitätsgrenzen hinaus die Erstellung multimedialen Lehrmaterials deutlich fördert.

Basierend auf früherer Erfahrung, unter anderem aus dem HyNoDe-Projekt, wurden folgende Ziele erreicht:

- *Personalisierung*:
  Es kann nicht erwartet werden, dass zwei Benutzer des Systems identische Ansätze und Ziele der Lehre verfolgen und somit identische Vorträge halten können. Vielmehr ist zur erwarten, dass Lehrende Bausteine gemeinsam nutzen, diese aber individuell einsetzen. Daraus entstand die Notwendigkeit zur Personalisierung und Versionierung des Materials, sowie zur persönlich gefilterten Benachrichtigung über interessante Änderungen im Material eines Kollegen.
- *Medien-Integration*:
  die Nutzung von Audio und Video in Vorlesungen wird bislang nicht nur durch die aufwendige Erstellung behindert. Ein weiteres Problem ist das Fehlen einer Infrastruktur für Speicherung, Transport, Empfang und Abspielen solcher Daten, die die folgenden Kriterien erfüllt. Zuallererst müssen die Medien in der in Vorlesungen oft erwünschten sehr guten Qualität zur Verfügung gestellt werden können.
  Ein wichtiger Punkt bezüglich Medienintegration war die Forderung nach dem Konzept *alternativer Medien*: das System wählt je nach Anforderung und aktueller Möglichkeit die passende Medienrepräsentation eines Inhaltes aus. Für diesen Punkt sind auch Benutzervorgaben entsprechend der oben genannte Personalisierung relevant [32].
- *Stabilität des Systems*:
  Das System arbeitet dezentral,um Stabilitätsprobleme von Netzen und Rechnern weitgehend abfangen zu können. Dies betrifft die eingesetzten Mechanismen zur Verteilung und zur Datenhaltung. Es betrifft aber auch das erstellte Material selbst, das dem System bei geeigneter Erstellung die automatische Auswahl aus mehreren Alternativinhalten ermöglicht und somit Probleme der Präsentationsgeräte umgeht (z.B. mangelnde Farbtiefe, mangelnde Rechenleistung oder fehlender Netzanschluß).
- *Copyright-Schutz*:
  Da aufgrund der finanziellen Situation von Hochschulen die Möglichkeit bestehen soll, eine kostenlose Nutzung von Lehrmaterial für außeruniversitäre Ausbildung auszuschließen, wurden gegenüber etwa normalen Dateisystemen erweiterte Mechanismen für den Zugriffsschutz benötigt.

Wegen der Nebenbedingung, die Infrastruktur auf den an Universitäten vorhandenen Systemen benutzbar bzw. portierbar zu halten, wurde das Projekt Medianode so ausgerichtet, eine eigene Infrastruktur zu schaffen, deren Programmquellen offen liegen und die streng konform zu den entsprechenden weltweiten Standards ist, um die vorgenannten Punkte zu adressieren.

2.2 Die Arbeitsziele

Es gibt viele erfolgreiche Ansätze und Arbeiten, die Applikationen mit der Fähigkeit ausgestattet haben, multimediale Inhalte zu verarbeiten, vorzuhalten und zu transportieren. Genauso ist es ein weitgehend verstandenes Problem, etwa im Bereich der Computer-unterstützten Softwareentwicklung (computer aided software engineering, *CASE*) oder von Computer-unterstützte Gruppenarbeit (computer supported collaborative work, *CSCW*), die Arbeit vieler Autoren zu integrieren.

Ziel dieser Arbeit ist eine Untersuchung der Hypothese, dass die Kombination der Fähigkeit zur Verarbeitung multimedialer Inhalte mit der Fähigkeit, die Arbeit vieler Autoren zu unterstützen und zu integrieren, Fragen aufwirft, deren Beantwortung noch aussteht.

Weiterhin wird die Arbeit zur Lösung entsprechender offener Fragen beitragen. Wir werden dazu im weiteren versuchen, die Anforderungen der weiter oben vorgestellten Beispiele zu erfüllen, wo möglich durch einfachen Transfer bestehender Techniken, vor allem aus den verwandten Bereichen Multimedia und kooperatives Arbeiten.

Aus den Beispielen aus den vorigen Abschnitten lassen sich die folgenden geforderten Systemeigenschaften identifizieren:

Unterstützung vieler Autoren

- Das System muss ein Konzept individueller Autoren besitzen, zum Beispiel durch Authentifizierung und Autorisierung und durch Benutzerprofile.
- Ein wichtiger Synergieeffekt ist die Wiederverwendung der multimedialen Inhalte. Dazu muss das System entsprechende Methoden zur Suche und Referenzierung anbieten.
- Um ein System zu realisieren, das mit der Zahl der Autoren skaliert, muss es die Systemverwaltung wie auch die Verwaltung der beteiligten Autoren entsprechend unterstützen.

Viele Autoren und Multimedia

- Die Koordination vieler Autoren erfordert die Verwaltung und Verwendung von Zugriffsrechten, welche bei multimedialen Inhalten an deren reiche Struktur angepasst sein müssen.

Rahmenbedingungen großer verteilter MM-Projekte

- Flexibilität, insbesondere bei der Unterstützung verschiedener Plattformen, Protokolle und Medienformate.
- Konfigurierbarkeit und Erweiterbarkeit: neue Fähigkeiten sollten einfach und damit möglichst dynamisch ins System integrierbar sein. Notwendig ist damit die (mögliche) Kapselung neuer Fähigkeiten in Einheiten, zuladbare Komponenten.
- Reservierung von (Lese-) Zugriffen, höhere Verfügbarkeit, als sie etwa das Internet bietet.
- Die Unterstützung der Repräsentation von Inhalten durch alternative Medien.

Flexibilität und Anwendbarkeit des Ansatzes

- Notwendig sind klar definierte, einheitliche, wiederverwendbare Schnittstellen, insbesondere für den Zugriff auf die multimedialen Inhalte und deren Struktur.
- Um verschiedene Medienformate mit ihren Systemanforderungen sinnvoll verarbeiten zu können, müssen reflektive Daten über das Zielsystem selbst im System zugänglich sein. Desweiteren sollte das Format dieser reflektiven Daten möglichst weit kompatibel mit den (Meta-) Daten über die multimedialen Inhalte sein, um etwa benötigte und vorhandene Bandbreiten vergleichen zu können.

2.3 Abgrenzung der Arbeit

Neben den oben gegebenen Zielen, die vor allem systemweite und statische Eigenschaften eines Systems betreffen, gibt es weitere Herausforderungen, die bei der Realisierung verteilter multimedialer Systeme mit vielen Autoren bestehen. Deren Untersuchung ist jedoch nicht Ziel dieser Arbeit. Der folgende Abschnitt soll die identifizierten Ziele gegen benachbarte Aspekte abgrenzen, welche nicht im Fokus der weiteren Betrachtungen liegen werden.

Inkonsistenzen des Systemzustandes

Im weiteren Fortgang werden Dienste definiert, die notwendig und hilfreich sind für Systeme, die multimediale Ergebnisse vieler Autoren sinnvoll integrieren sollen. Für die Realisierung dieser Dienste wird zunächst eine ideale Verteilung des Systemzustandes vorrausgesetzt. Eine Integration dieser Lösungen mit Konzepten zur Behandlung von Inkonsistenzen im verteilten Systemzustand ist Gegenstand anderer Arbeiten [10], [39].

Gleichzeitiges Ändern eines Monomediums

Gleichzeitiges Editieren desselben Medieninhaltes wird in dieser Arbeit nicht betrachtet. Diese sehr nützliche Funktionalität ist zum einen ein intensiv behandelter Gegenstand in den Arbeiten zu CSCW, der über den Rahmen dieser Arbeit hinausgeht.

In den im weiteren vorgestellten Lösungen wird paralleles Ändern desselben Inhalts durch Sperrmechanismen verhindert, also durch das Setzen von Kennzeichen, die schreibenden Zugriff für weitere Autoren während einer Änderung verbieten.

Gleichzeitiges Editieren wird vor allem durch medientypspezifische, spezielle Mehrbenutzerprogramme implementiert, die ohne weitere Integrationsarbeiten mit den vorgestellten Lösungen kombiniert werden können.

Workflow

Die dynamischen Aspekte von Workflow, also die Verfolgung von Problemen, von deren Status und die dynamische Zuweisung und Aktivierung entsprechender Rollen und Aktivitäten zu Personen, sind nicht Gegenstand dieser Arbeit.

Diese Arbeit zielt auf die Verwirklichung statischer Aspekte, die gemeinsames Arbeiten ermöglichen (etwa Organisation der Daten, notwendige Architektur), und die auch eine Basis für die Realisierung der nützlichen Workflow-Funktionalitäten bieten sollen. Sie zielt nicht direkt auf die Integration oder Realisisierung von Workflow für verteilte Multimediaanwendungen selbst.

Im folgenden werden dennoch zwei Mechanismen entwickelt oder angepasst, die auch Gegenstand der Arbeiten an Workflow sind: aktivitätsspezifische Rollen in Kapitel 6 und Schemata der Benachrichtigung bei Änderungen der Systeminhalte in Kapitel 7. Workflow jedoch nutzt unter anderem solche Mechanismen und fügt die aktive Verfolgung von Problemen, Arbeiten und Zuständen durch das System hinzu. Diese aktive Dynamik ist nicht Ziel dieser Arbeit.

Kapitel 3 - Analyse des Umfelds

Die Abschnitte in diesem Kapitel betrachten den Stand der Technik und Forschung bezüglich verteilter, multimedialer Systeme. Mit dem Fokus auf der Erstellung der Inhalte wird hier insbesondere deren Eignung bzw. beschränkte Eignung zur Erstellung verteilter multimedialer Anwendungen mit vielen Autoren diskutiert. Es werden vorhandene Ansätze daraufhin untersucht, wieweit mit deren Lösungen Applikationen befähigt werden können, mit vielen Autoren interagieren zu können

Neben Forschungsarbeiten existieren Standards und Techniken aus den Bereichen Middleware, WWW, Multimedia-Formate, Versionsverwaltung in der Programmerstellung, Datenbanken, Dateisysteme und Computer-unterstützte Gruppenarbeit (CSCW), die miteinander das meiste der Problemfelder abdecken.

Das Ziel dieses Kapitels ist es, die Problemfelder zu identifizieren, in denen die betrachteten Techniken nicht hinreichen, um die erwünschten System-Eigenschaften zu realisieren. In den weiteren Abschnitten dieses Kapitels werden dazu zunächst Techniken verteilter multimedialer Applikationen betrachtet. Dann werden Techniken aus den Bereichen Rechte- und Versionsmanagement, die bei der Koordinierung der Arbeit mehrerer oder vieler Autoren zum Einsatz kommen, besonders im Hinblick auf multimediale Anwendungen untersucht.

3.1 Verteilte Multimedia-Applikationen

Verteilte Multimedia-Applikationen stellen hohe Anforderungen in einigen technischen Gebieten. Diese beinhalten unter anderem Netzwerktechnik, Synchronisation des Transports und der Ausgabe von Medienströmen und Gruppen von Medienströmen, aber auch Softwarearchitektur für komplexe, notwendigerweise flexibel zu wartende und erweiterbare Systeme.

Der vor allem in der vergangenen Dekade von Industrie und Forschung etablierte Grundstock an Ergebnissen - vor allem Standards und etablierte wissenschaftliche Konzepte - bietet hier eine notwendige und für viele Aufgabenstellungen sehr weitreichende Grundlage zur Lösung.

Die vorhandenen Ansätze können so eingeteilt werden:

- Gesamtsysteme, bestehend aus Frameworks mit Klassenbibliotheken und Komponenten (meist Forschungssysteme)
- Ansätze, die aus der Standardisierung von Medien-Datenformaten hervorgegangen sind (etwa MPEG)
- das WWW als de-facto-Standardisierung von Hypertext-Format und -Transport (HTML, HTTP, aber auch WebDAV, XML)
- Ansätze, die aus der Standardisierung von Middleware und Komponentenschnittstellen hervorgegangen sind
  (CORBA, ActiveX, RPC, OpenDoc, JavaBeans, Swing)

Im folgenden werden diese Ansätze vorgestellt, teilweise anhand jeweils typischer oder wichtiger Beispiele.

3.1.1 Prototypen

Aus der Forschung in der Form von Frameworks und generischen Komponenten vorgeschlagene Gesamtsysteme bieten die Möglichkeit, benötigte neue Komponenten, etwa spezielle Protokolle, effizient zu realisieren und eine gewünschte verteilte Multimedia-Applikation möglichst flexibel komponieren. Beispiele hierfür sind das an der Universität Stuttgart realisierte System CINEMA [42] und Vorschläge einiger Dissertationen, etwa [5], [30]. Kommerzielles Beispiel hierfür ist z.B. auch JMF [97], auch Apples Quicktime [127] fällt grob in diese Kategorie.

Diese Frameworks bestehen, wie exemplarisch auch CINEMA, neben einem Toolkit für Aufgaben wie Setup, Logging und Debugging, vor allem aus einer Sammlung von Komponenten, die Ströme von Multimedia-Daten definieren, aufsetzen, aktivieren, rekonfigurieren und synchronisieren sollen. Diese so genannten Stream-Handler-Architekturen adressieren systemnahe Probleme, die so noch nicht durch den Stand der Technik, etwa in Middleware-Standards oder in MPEG-4, gelöst sind. Einige aktuelle Entwicklungen, zu Quality of Service (*QoS)* etwa in CORBA [131] und zu Strömen mit variabler Bitrate in MPEG-4 [23], sind aus den Ergebnissen solcher Forschungsarbeiten entstanden.

Diese Arbeit hat als Rahmenbedingung, multimediale Inhalte zu unterstützen. Der Stand der Technik hierzu, etwa MPEG-4, muss mit dem Ergebnis dieser Arbeit genutzt werden können. Der Fokus dieser Arbeit aber liegt auf einer abstrakteren, applikationsnäheren Ebene: Wie muss die Darstellung, Organisation und Behandlung der Inhalte einer Applikation modifiziert werden, wenn die Applikation bei der verteilten Erstellung ihrer Inhalte unterstützt werden soll?

Wegen dieses Schwerpunktes wird an dieser Stelle nur auf die Weiterentwicklung bei der QoS-beachtenden Präsentation und der Synchronisation von multimedialen Daten in dedizierten, prototypischen Frameworks hingewiesen und nicht näher eingegangen. MPEG-4 wird im Abschnitt 3.1.2 kurz vorgestellt, und die Unterstützung multimedialer Charakteristiken von verteilten Applikationen durch MPEG-4 reicht in diesem Zusammenhang für die Diskussion des in dieser Arbeit behandelten Problemfeldes.

3.1.2 MPEG-4: Mehr als ein Medienformat

Ausgehend von Arbeiten an der physikalischen Darstellung, am Format multimedialer Inhalte, entwickelte sich aus den MPEG-Standards ein Framework ähnlich den kurz angesprochenen Systemen aus Abschnitt 3.1.1. Im Unterschied zu diesen aber steht der Umfang der MPEG-Spezifikationen, vor allem aber die Zielrichtung: MPEG will eine Standard-Lösung, die weitestgehende Interoperabilität zwischen Komponenten verschiedener Hersteller ermöglicht. Die Genauigkeit und allgemeine Anwendbarkeit der Spezifikationen ist hier viel wichtiger als eine Abdeckung des in speziellen Prototypen technisch Machbaren.

Aufbauend auf den ISO-Standards MPEG-1 [21] und MPEG-2 [22] entwickelte die Motion Pictures Expert Group ab 1993 MPEG-4 [23]. MPEG-4 übernahm von seinen Vorgängern die sehr detaillierte und eindeutige Art der Spezifikation für Datenformat und Transport von Video- Audiodaten. Sehr großen Wert legt das MPEG-Standardgremium auf Interoperabilität, weswegen die MPEG-Standards auch auf sehr systemnahem Level Vorgaben machen, etwa in Form von vielen sogenannten *Profiles* über die in MPEG möglichen Medienformate. Die MPEG-Standards machen Vorgaben nur über testbare Schnittstellen, etwa Parsern und Protokollmaschinen, nicht über deren Implementierungen.

Einen großen Teil des Standards MPEG-4 bildet die Spezifikation des MPEG-4-*Terminals*. Dieses abstrakte MPEG-4-Endgerät bietet dedizierte Kanäle für Benutzer-Feedback, allerdings auf der erwähnten systemnahen Ebene. Eine Applikation muss, wenn gebraucht, applikationsrelevante Semantik von Feedback selbst generieren, indem sie erhaltenes Feedback selbst analysiert, aggregiert und mit ihrer Applikationssemantik besetzt. Da Endgeräte, die Autorensysteme sind, zu einem großen Teil systemferne Information vom Benutzer sammeln, bietet das MPEG-4-Terminal hier wenig Unterstützung.

Hier ist vor allem interessant, dass der Standard MPEG-4, über MPEG-2 hinausgehend die Integration vieler (etwa auch generierter) Inhaltsströme zulässt und den Aufbau komplexer Szenarien mit vielen Inhaltsquellen erlaubt. Dazu spezifiziert MPEG-4 neben dem reinen MPEG-4-Terminal eine Transportschnittstelle (*DMIF*, delivery multimedia integration framework) und ein Session- und Resource Management, *SRM* (Abbildung 2).

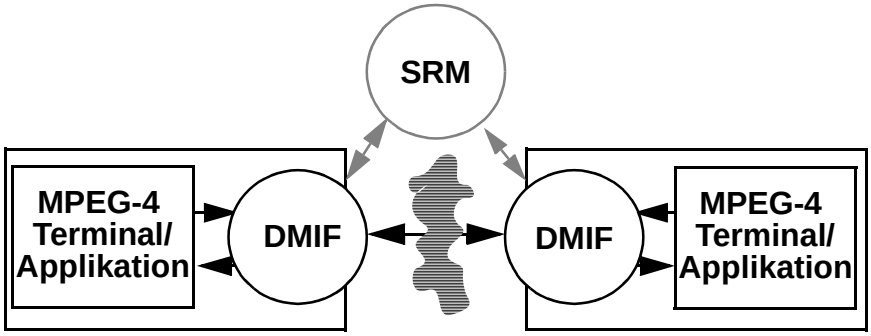

Abbildung 2: eine MPEG-4 Session mit zwei MPEG-4-Terminals

Weiterhin ist für die Inhalte multimedialer Applikationen deren zusammengesetzter Aufbau und ihre vielfältigen semantischen internen Beziehungen kennzeichnend. Hier bietet MPEG-4 zum einen die Möglichkeit komplexer Szenenbeschreibungen, die sich aus Objekten zusam-

mensetzen können (siehe Abbildung 3), zum anderen unterstützt es mit dem erweiternden Metadatenstandard MPEG-7 die Abbildung von Inhaltssemantik auf Metadaten.

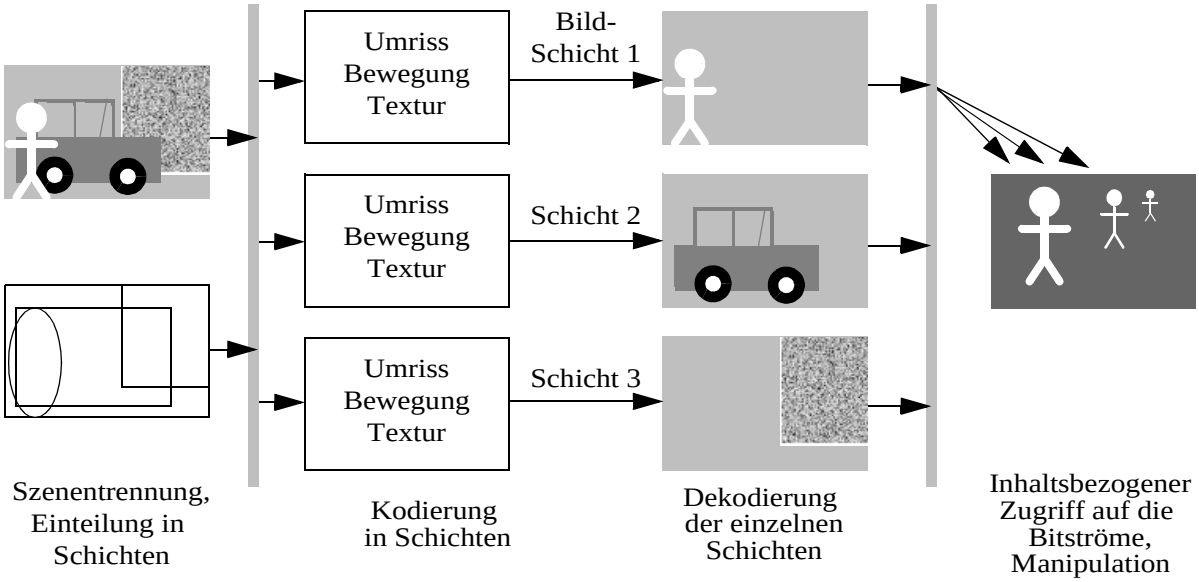

Abbildung 3: Beispiel für die Zusammensetzung einer MPEG-4-Szene aus Audio-Visuellen-Objekten (*AVO*)

MPEG-7

Mit MPEG-7 will die ISO einen Standard für Metadaten etablieren, die zusammen mit den Inhaltsdaten zum Beispiel von MPEG-4-Systemen gespeichert, transportiert oder genutzt werden können [23].

Dazu spezifiziert MPEG-7 einen auf XML basierenden Mechanismus um themenbezogene, so genannte *Schemes* zu definieren (etwa für Informationen, die für einen bestimmten Industriezweig interessant sind, wie etwa Filmindustrie oder Lehreinrichtungen) und definiert selbst einige Schemes, zum Beispiel eines für die grundlegende Beschreibung des Formats, der Sprache etc. von Datenströmen.

Neben der Kodierung von Metadaten definiert MPEG-7 noch die Schnittstellen, um diese Metadaten zu erzeugen oder ein Medium zu annotieren (für merkmalsextrahierende Werkzeuge) und um die Merkmale abzufragen.

Bewertung

Tabelle 1: Bewertung von MPEG-4 als Grundlage verteilter MM-Autorensysteme

Positiv	Negativ
+ Unterstützung von Medienströmen (Adaption, Synchronisation)	- keine Generierung von Applikations-Level-Feedback
+ Unterstützung semantischer Aspekte (Metadaten, MPEG-7)	
+ Unterstützung von Sitzungsaufbau und -Anpassung auch komplexer, verteilter Multi-Party-Szenarios	- eingeschränkte Dynamik der Inhalte
+ hohe Interoperabilität der MPEG-Implementierungen (noch etwas fraglich)	- keine schnelle/ad-hoc-Erweiterbarkeit

3.1.3 WWW: Mehr als Hypertext

Tim Berners-Lee griff bekannte “Hypertext”-Ideen auf, als er die Grundlagen für das heutige WWW legte. Einerseits sollten Informationen direkt und plattformunabhängig gezeigt und andererseits Verknüpfungen *(Links)* zu anderen Informationen, die sich im Netzwerk befinden, erstellt werden können: dazu verwendete er die Hypertext Markup Language HTML mit einer SGML-ähnlichen Syntax, die recht schnell um multimediale Eigenschaften erweitert wurde, etwa sehr bald um das Kennwort *<img>*, um Bilder in die Dokumente mit einzubinden. 1990 wurde der Name für den neuen Dienst im vorhandenen Netzwerk festgelegt: World Wide Web, WWW. 1991 wurde das WWW-Protokoll-Pendant HTTP (Hypertext Transfer Protocol) in der Version 0.9 implementiert. Der anschließende schnelle Erfolg des WWW hatte große Nachteile für dessen technische Entwicklung: neue Ausgaben der Standards wurden abwärtskompatibel, auch unter Beibehaltung von Fehlentscheidungen, gehalten, voneinander abweichende Implementierungen wurden bei folgenden Revisionen der Standards berücksichtigt [72].

Da bereits bei der Verabschiedung davon ausgegangen wurde, dass HTTP/1.0 schon bald durch eine neuere Version abgelöst wird ([72], S. 1), wurde mehr Wert auf Einfachheit als etwa auf Effizienz oder langfristige, etwa modulare, Erweiterbarkeit gelegt. Die Ziele des noch aktuellen HTTP/1.1 deckten sich mit denen von HTTP/1.0, wurden aber um die Eliminierung der Probleme von HTTP/1.0 erweitert. Besonderes Augenmerk wurde auf eine ressourcenschonende Mehrfachverwendung offener Netz-Verbindungen und auf die Unterstützung von Zwischenspeichern im Netz, Caches gelegt.

Die Funktion von HTTP hängt in seiner Einfachheit von einem “sicheren” Übertragungsprotokoll ab. Im Kontext des Internet ist dies praktisch immer das Transfer Control Protocol *TCP* über dem Internet-Protokoll *IP*. Um eine sichere Übertragung zu gewährleisten, benutzt TCP verschiedene Techniken, die teilweise schlecht mit HTTP harmonieren. So benutzten die

ersten HTTP-Versionen eine neue Verbindung für jedes neu angefragte Objekt. Dies wird immer noch von den Servern unterstützt und auch von vielen Clients so gehandhabt.

TCP implementiert einen TIME_WAIT Status, der besagt, dass ein Server nach dem eigentlichen Schließen einer TCP-Verbindung Informationen dieser Verbindung für eine bestimmte Zeit speichern muss, um verspätete Pakete richtig behandeln zu können. Die Standardeinstellung für diesen TIME_WAIT beträgt 240s [122], was bei einem häufig frequentierten WEB-Server zu sehr vielen Verbindungen im TIME_WAIT-Status und damit einem hohen Ressourcenverbrauch auf dem Server führt.

TCP benutzt beim Verbindungsaufbau den sogenannten “Three-Way-Handshake”, das bedeutet, dass Nutzdaten erst nach 2 RTTs (Round Trip Times) gesendet werden können. Diese Zeit geht natürlich bei jedem erneuten Verbindungsaufbau verloren.

Ein Problem, das aus der Nutzung von TCP, aber auch aus der Problematik resultiert, dass eine Verbindung pro URI geöffnet wird, ist, dass ein typischer HTTP-Request nur aus wenigen tausend Bytes besteht. Dies ist ungünstig, da TCP eine Technik zum Verhindern von Pufferüberläufen und Überlastungen benutzt, die auch unter dem Namen “Slow Start” bekannt ist.

Dabei wird beim Beginn der Übertragung als Paketgröße eine von allen Netzwerken entlang der Request/Response-Kette handhabbare Fenstergröße benutzt. Diese beträgt in reinen Ethernet-Netzwerken 1460 Bytes, im Internet beträgt die Start-Fenstergröße 536, respektive 512 Bytes [122]. Gelingt diese Übertragung ohne dass eine bestimmte Zeit überschritten wird oder ein Pufferüberlauf gemeldet wird, so wird das nächste Paket mit der doppelten letzten Fenstergröße (Congestion-Window-Size) gesendet. Gelingt auch diese Übertragung, so wird wiederum verdoppelt, so dass nach einem langsamen Start eine exponentielle Übertragungsratensteigerung erreicht wird. Die maximale Größe beträgt bei TCP inklusiver Header 65535 Bytes (64KB). Sollte eine Übertragung nicht gelingen, ist also die Leitung oder der Puffer des Empfängers ausgelastet, so wird mit der Fenstergröße der letzten geglückten Übertragung weitergesendet ([52], S. 538 ff.).

Aus der Initialgröße von 536 Bytes läßt sich schon schließen, dass bei einem “normalen” Request allein durch die Header-Informationen diese Größe erreicht und übertroffen wird. Somit muss wiederum eine RTT gewartet werden, bis nur der Request gesendet wurde [122]. Der Server sendet nun die Antwort ebenfalls mit einer geringen “Congestion-Window-Size”, so dass auch die Antwort häufig mehrere RTTs beansprucht (bei einer Antwortgröße ab 2KB).

Da HTTP/1.0 ein zustandsloses Protokoll ist, gehen alle Informationen, die über eine TCP-Verbindung bekannt sind, bei deren Abbau verloren. Außerdem können mehrere TCP-Verbindungen zu einem Server keine Informationen über Überlastung untereinander austauschen. So ist es für nachfolgende Verbindungen unmöglich, von eventuellen Überlastungen oder Routerausfällen zu erfahren. Jede neue Verbindung zu dem gleichen Host verfügt also über keine Informationen über eventuelle Überlastungsprobleme. Jede Verbindung muss also ihre “Erfahrungen” selbst machen, was ineffizient ist.

Weitere Probleme

Es wurde ersichtlich, dass die herkömmlichen HTTP Implementierungen verschwenderisch mit den Internet-Ressourcen umgehen. Dies führte zu Überlastungen und Teilausfällen von

wichtigen Verbindungen. Leider ist auch eine Möglichkeit den Verkehr zu begrenzen nur rudimentär in HTTP/1.0 enthalten, nämlich die des Caching.

So wird der in RfC1945 spezifizierte generelle Pragma-Header zur Steuerung des Caching (*no cache*) in der Praxis nur von den wenigsten Cache-Servern berücksichtigt [117], die Möglichkeiten für hierarchisches Caching fehlen ganz.

Ein weiterer Kritikpunkt ist die nur sehr einfache Sicherheit, die HTTP/1.0 bietet. Die Benutzerinformationen und sogar das Paßwort werden in der Standardkonfiguration im Klartext übertragen. Sicherheit sollte nicht in HTTP selbst implementiert werden, sondern in anderen darüber oder darunterliegenden Schichten. Die schwache Sicherheit ist der Grund dafür, dass Techniken wie z.B. SSL (Secure Socket Layer) zusätzlich zu HTTP benutzt werden.

Ferner unterstützt HTTP/1.0 kein “virtuelles Hosting”, da nach erfolgtem Verbindungsaufbau zum Zielserver nur die relative URI übergeben wird und nicht zusätzlich der Host-Name des Zielservers. Das bedeutet, dass für jede Domäne im Internet eine IP-Adresse benötigt wird. Dies führte zu Befürchtungen, dass der Adressraum den IP bietet irgendwann erschöpft sein wird.
Dadurch erhielten die Bemühungen eine neue Version des Internet-Protokolls IP zu entwickeln (IPv6), einen Aufschwung. Außerdem mussten WWW-Server, die mehrere Domänen beheimaten, auch mit mehreren IP-Adressen konfiguriert werden, was höhere Ressourcenauslastung nach sich zog.

Für HTTP/1.0 wurden die “Internet Media Types” von MIME übernommen, wodurch eine einheitliche Repräsentation der Inhaltstypen festgelegt wurde. Ein Problem stellt hingegen die “angepasste” Implementierung von MIME in HTTP/1.0 dar. Hier wurden einige Abweichungen in Kauf genommen, um eine besonders gutes Ergebnis für WWW-Verkehr zu erzielen.

So kennt bespielsweise RfC1521 (MIME) kein “Content-Encoding”-Header-Feld, RfC1945 (HTTP/1.0) hingegen kein “Content-Transfer-Encoding”-Header-Feld [72]. Aus diesem Grund müssen Proxies, bzw. Gateways von HTTP zu MIME-kompatiblen Protokollen Anpassungen vornehmen.

Da die HTTP-Implementierung von MIME nicht genau der MIME-Spezifikation folgt, sollte das optionale MIME-Version General-Feld nicht benutzt werden.

Keine Neuerung ohne Probleme - so auch bei HTTP/1.1. Was sicher als erstes auffällt, ist, dass von der Einfachheit der ursprünglichen HTTP Implementationen HTTP/0.9 und HTTP/1.0 nicht mehr viel übrig geblieben ist. HTTP/1.1 ist ein schwieriges, komplexes Protokoll geworden. Dies liegt nicht zuletzt daran, dass trotz neuer Konzepte und Funktionen die Kompatibilität zu den Vorgängerversionen gewahrt bleiben sollte. Ebenso wie diese besteht HTTP/1.1 aus einem großen Protokoll, das nur schwer erweitert werden kann. So sind Erweiterungen nur in der Form möglich, dass sie das gesamte HTTP benutzen. Es ist leider nicht möglich, nur Teile davon, wie z.B. die “Persistent Connections” als Transportschicht zu nutzen.

Ein Nachteil, der bereits in HTTP/1.0 bestand, dort aber aufgrund des Konzepts und der anderen Mängel kaum ins Gewicht fiel, beruht darauf, dass die HTTP-Steuerung auf dem US-ASCII-Kode beruht, um einfaches Verständnis und einfache Fehlersuche zu ermöglichen. Leider ist dies ineffizient, da die gebotene Funktionalität durch geeignete Kodierung mit sehr viel

weniger Bytes möglich wäre. Dies wirkt sich besonders auf Verbindungen mit geringer Bandbreite, wie z.B. kabellose Verbindungen, negativ aus ([88], S. 6, [123] und [76]).

Ein weiteres, allerdings schon bekanntes Problem, stellt die Abweichung vom MIME-Standard für die HTTP-Übertragung dar. Diese Problematik wurde durch ein neues, wiederum abweichendes Header-Feld (*Transfer-Encoding*) noch verstärkt.

Ebenfalls problematisch ist, dass nur einige Implementationen HTTP sin einer gesamten Komplität berücksichtigen, es aber bei geringen Abweichungen schon dazu kommt, dass viele Verbesserungen, z.B. das Caching nicht mehr funktionieren, bzw. nicht mehr benutzt werden dürfen. Dieser Punkt wird noch verschärft, wenn man bedenkt, dass der Prozess, der Entwicklung von HTTP/1.1 zu RfC2616 über vier Jahre in Anspruch nahm [65]. Dabei wurden natürlich auch Änderungen an der Spezifikation vorgenommen. Viele Hersteller und Entwickler wollten oder konnten nicht warten, bis ein endgültiger Standard verabschiedet wurde, so dass sie ihre Software kompatibel zu den zu diesem Zeitpunkt gültigen “Standards” entwickelten. Ein spezielles Problem ergibt sich somit daraus, dass man nur anhand der Versionsnummer “1.1”, nicht auf die eigentliche Implementierungsform schließen kann.

Ebenfalls wenig hat sich an der reinen Sicherheitsfunktionalität von HTTP/1.1 geändert, auch hier muss auf zusätzliche Module oder auf außerhalb von HTTP gelegenen Mechanismen (s.o.) zurückgegriffen werden, um nutzbare Sicherheit zu erreichen. Allerdings ist zu anzumerken, dass eine neue Authentifizierunstechnik bewußt parallel zu HTTP/1.1 entwicklet wurde [96], [65].

Ein weiteres Problem, welches auf dem monolitischen Aufbau von HTTP und den vielfältigen Erweiterungen beruht, ist, dass die Kommunikation von Zusatzapplikationen mittels HTTP-Möglichkeiten getunnelt werden muss (man kann dazu die POST-Methode benutzen) und so der Inhalt nicht von Firewalls identifiziert werden kann. Deshalb ist es den Firewalls nicht möglich, ihre Sicherheitsrestriktionen anzuwenden ([88], S. 2 ff.). Außerdem wird eine große Zahl von Kodierungs-Operationen notwendig, die lediglich im Aufbau von HTTP begründet sind.

TCP-Probleme

Auch wenn in RfC2616 wiederum explizit darauf hingewiesen wird, dass TCP/IP nicht als ausschließliches Übertragungsprotokoll genutzt werden muss, so sind doch noch keine weiteren Implementierungen bekannt. Die Nutzung von TCP als Standardprotokoll wird noch unterstrichen durch die bessere Anpassung von HTTP/1.1 an TCP-Eigenheiten, wie dies z.B. bei den “Persistent Connections” geschehen ist. Leider treten aber auch bei HTTP/1.1 einige Probleme mit TCP als Übertragungsprotokoll auf.

Ein Problem, welches auf der Einführung des "Pipelining" beruht, ist die standardmäßige Nutzung des Nagle-Algorithmus durch TCP. Dieser verhindert das Senden von sehr kleinen Paketen, wenn die Information nur sehr langsam zum TCP/IP-Stack fließt. Ursprünglich wurde dieser Algorithmus für Anwendungen wie *Telnet* oder *rlogin* benutzt, da hier der Benutzer langsam (für Computer) nacheinander Buchstaben über die Tastatur eingibt und diese in der ursprünglichen TCP/IP-Implementierung direkt versandt wurden. Oftmals nahm damals der Header mehr Platz in Anspruch als die eigentlichen Nutzdaten.

TCP mit Nagle-Algorithmus sendet zunächst das erste Byte, sammelt dann alle anderen ankommenden Informationen,bis die Bestätigung (ACK) über die geglückte Übertragung des ersten Byte eingetroffen ist. Dadurch werden gezielt Übertragungen verhindert in denen der Overhead ein vielfaches des eigentlichen Datenaufkommens ausmacht. Nagles-Algorithmus ist in den meisten TCP-Implementierungen realisiert ([52], S. 534 ff.).

Experimente haben gezeigt, dass der Nagle-Algorithmus sich negativ auf die Perfomance von HTTP/1.1 auswirkt, insbesondere das "Pipelining" führt zu diesen Problemen [18], [64]. Es wird daher empfohlen auf Web-Servern und Clients den Nagle-Algorithmus mittels der TCP_NODELAY-Socket-Option abzuschalten [64].

Ein grundsätzlliches Problem des "Pipelining" besteht in seinem sequentiellen Charakter. Der Server kann auf nacheinander gesendete Requests nur in der gleichen Reihenfolge antworten und nicht in der, in der er die Anfragen abgearbeitet hätte. Außerdem ist es nicht möglich, mehrere Objekte gleichzeitig zu senden oder zu empfangen (Multiplexen). Dadurch wird den User-Agents die Webseitendarstellung erschwert, da sie die Objekte in den Webseiten nicht gleichzeitig aufbauen können. “Intelligente” User-Agents können dieses Problem durch die Anfrage von kleineren Teilen der Objekte (“Zeitmultiplex”) mindern.

Die Gerechtigkeitsproblematik besteht in HTTP/1.1 ebenfalls weiter. Auch wenn “Persistent Connections” und “Pipelining” benutzt werden, so hindert (noch) keiner einen User-Agent daran, gleichzeitig mehrere TCP-Verbindungen zu benutzen und damit andere User-Agents zu benachteiligen. Werden mehrere TCP-Verbindungen benutzt, besteht auch ein weiteres Problem: die Verbindungen tauschen untereinander keine Überlastungsinformationen aus. Dieses Verhalten könnte durch eine Erweiterung von TCP (Sharing of TCP-Control-Blocks) behoben werden. Vgl. [119].

Empfohlen werden maximal zwei gleichzeitige TCP-Verbindungen zu einem Host, allerdings wird erst der Einsatz von weiteren Techniken wie RED (Random Early Detection) auf der Server-Seite zu einer garantiert gerechten Ressourcen- und Bandbreitenverteilung führen [85].

Der folgende Punkt kann nicht unbedingt als Problem, wohl aber als kosmetischer Fehler der “Persistent Connections” bezeichnet werden. Zwar werden durch die “Persistent Connections” die Nachteile des “Slow Start”-Algorithmus und des Verbindungsaufbaus von TCP gemindert und schlagen sich in einer starken Reduzierung der transportierten Pakete nieder [64], allerdings fallen die Vorteile für den Endbenutzer, für den nur die Zeit, die der Seitenaufbau benötigt relevant ist, geringer aus. So treten die Vorteile der “Persistent Connections” besonders bei schnellen Verbindungen zu Tage, bei Modem-Verbindungen wird eine Verbesserung von lediglich 11%, bei ISDN-Verbindungen von 27% erreicht [94][95]. Ihre ganzen Vorteile können die “Persistent Connections” erst in Verbindung mit “Pipelining” und Kompression ausspielen.

3.1.3.1 HTTP/NG

Noch vor den Bemühungen im Jahr 1995, HTTP/1.1 zum Standard zu führen, dachte man daran, das problematische HTTP/1.0 durch ein komplett neues Protokoll zu ersetzen [123]. Dieser *HTTP-NG* (HTTP-NextGeneration) genannte Vorschlag sollte größtenteils auf Rück-

wärtskompatibilät verzichten, dafür in der Übergangsphase lediglich zwischen Proxies Anwendung finden (diese übersetzen zwischen HTTP/1.0 und HTTP-NG) [125], [76] und [106]. Aufgrund dessen wäre es möglich, ein veraltetes Konzept komplett zu überarbeiten.

Insbesondere sollten die noch vorhandenen Schwächen von HTTP/1.0 beseitigt werden. Hinzugekommen sind weiterhin die Möglichkeiten, das Protokoll als Grundlage für neue Techniken zu nutzen und es um diese erweitern zu können.

Ziele

Die ursprünglichen Ziele lauten wie folgt [123]:

- Einfachheit
- Leistungsfähigkeit
- Asynchronität (kein “Pipelining”, neue Requests, ohne auf die Berarbeitung eines vorigen Requests warten zu müssen)
- Sicherheit, Unterstützung für Verschlüsselung
- Authentifizierung aller an einer Transaktion beteiligten Parteien
- Erweiterbarkeit
- Charging, Unterstützung für Online-Bezahlung

Später wurde diese dann um weitere Ziele ergänzt:

- Effizienz [87]
- besondere Unterstützung für kabellose Verbindungen

HTTP/NG: Status

Die Entwicklung von HTTP-NG als Gesamtarchitektur endete im Dezember 1998 . Zunächst nach den Erfahrungen mit HTTP/1.0 1995 als möglicher Nachfolger vorgestellt, konzentrierte sich die Arbeit der Internet-Gemeinschaft, des W3C und der Industrie auf die Weiterentwicklung vorhandener Technik zu HTTP/1.1. Generell gibt es nur wenig Interesse an dieser Art Umstrukturierung des WWW-Transportprotokolls.

Das W3C zieht daraus die Konsequenzen, dass die Konzentration auf die Weiterentwicklung der Transportschicht und nicht die Promotion der gesamten Architektur liegen sollte. Das W3C will die Entwicklung des WWW in kleineren Schritten angehen [86]. Das WebMUX-Protokoll weist den höchsten Entwicklungsstand auf. Für die anderen Schichten existieren Working-Drafts, die abstrakt eine mögliche, zukünftige Funktionsweise beschreiben, für die aber bisher nur der Plan einer Testanordnung [112] existiert.

Insbesondere für den Remote Procedure Call existieren einige erfolgreiche Konkurrenzkonzepte, u.a. RPC, RPC II und seit neuerem SOAP.

3.1.3.2 HTML, XML, Metadaten

HTML ist das Hypertextformat des WWW. Nach einigen Entwicklungsstufen wurde diese Sprache durch die formalere und breitere Definition der Extensible Markup Language (*XML*)

abgelöst. Hierbei ist der Einsatz XML kein Ersatz für HTML, sondern eine evolutionäre Weiterentwicklung. Genauer gesagt, ist die aktuelle HTML-Version *XHTML* eine Anwendung der Spezifikation XML, Version 1.0, auf dem Gebiet Hypertext [126], die sich unter dieser Vorgabe möglichst nahe an die Spezifikation von HTML 4.0 anlehnt.

Das wesentliche Konzept von HTML ist unter dem Namen Generic Markup bekannt. Dies bezeichnet die Vorgehensweise, Texte mit Auszeichnungen zu versehen. Bei der üblichen Vorgehensweise, etwa bei Textverarbeitungen, werden Texte fast ausschließlich mit Auszeichnungen versehen, die der Formatierung dienen. Dazu gehören Attribute, die z. B. Einfluß auf den Schriftschnitt (kursiv oder aufrecht) nehmen, das Schriftgewicht (fett oder normal) bestimmen oder die Schriftfamilie auswählen. Diese Annotationen werden bei der Präsentation der annotierten Inhalte normalerweise nur genutzt, aber nicht selbst dargestellt. Da es hier um die Steuerung von visuellen Attributen geht, spricht man in diesem Fall auch von Visual Markup. Im Gegensatz dazu konzentriert sich das Generic Markup darauf, semantische Auszeichnungen in den Text einzubringen, die eine Aussage über die Bedeutung der markierten Textstelle machen. Anstatt beispielsweise Zitate als “kursiv” zu markieren, ist es sinnvoller, sie als “Zitat” zu markieren.

Ein wesentlicher Meilenstein in der Umsetzung dieser Ideen war die Entwicklung der Standard Generalized Markup Language (*SGML*). XML ist weitestgehend eine Untermenge von SGML, jedoch viel schlanker als der komplexe und dadurch schwer zu implementierende ISO-Standard SGML [79].

Neben der Grundidee Generic Markup ist ein wichtiges Konzept von XML der Dokumenttyp. Der Dokumenttyp beschreibt eine Klasse von Dokumenten, die sich in ihrem strukturellen Aufbau gleichen. Die formale Definition eines Dokumenttyps in XML muss die Flexibilität besitzen, die Gemeinsamkeiten einer Dokumentklasse festzulegen, aber zugleich genügend Spielraum für ein konkretes Dokument zu lassen. Die Dokumenttypdefinition muss also den Titel als zwingenden Bestandteil enthalten, die Anzahl der Kapitel aber offenlassen [79].

Ein XML- Dokumententyp definiert eine Baumstruktur, die einzelnen Elemente sind ineinander geschachtelt. XML bietet die Möglichkeit, solche Dokumenttypdefinitionen (abgekürzt *DTD*) für beliebige Arten von Dokumenten zu formulieren. Die bekannteste DTD ist die XHTML-DTD. Sie definiert die Hypertext Markup Language. HTML ist allerdings ein schlechtes Beispiel für die Umsetzung der Generic-Markup-Idee: HTML enthält nur wenige logische Auszeichnungen, dafür viele visuelle Auszeichnungen (z. B. “b” für bold und “i” für italic). Für die Gliederung eines Textes bietet HTML lediglich Überschriften, aber keine Elemente wie etwa “Kapitel” oder “Abschnitt”.

Welcher Dokumenttyp von HTML definiert wird, wird man nur schwer erkennen können: HTML-Dokumente sind Homepages, wissenschaftliche Abhandlungen, Kochrezepte, virtuelle Warenhäuser, usw. HTML-Dokumente sind keine Dokumente eines einzigen Typs. Ein wichtiger Vorteil von XML besteht darin, dass man einen Arbeitsschritt mit Bezug auf die DTD durchführt und dass das Ergebnis auf alle Dokumente dieses Typs angewendet werden kann. Zum Beispiel beziehen sich bei der Formatierung die Anweisungen zur Darstellung auf Elementnamen, die in der DTD deklariert werden. Eine einmal geschriebene Formatierungsfunktion kann dann für alle Dokumente, die gemäß dieser DTD ausgezeichnet sind, benutzt

werden. So kann es sich beispielsweise lohnen, eine DTD für eine Buchreihe oder das gesamte Programm eines Verlags zu entwerfen. Es wird dann einmal der Produktionsablauf für diese DTD implementiert und fortan können beliebig viele Manuskripte ohne Mehraufwand verarbeitet werden. Im Gegensatz dazu wäre eine DTD je Buch sinnlos und zu aufwendig. Eine weitere naheliegende Idee, einen inflationären Gebrauch von DTDs zu vermeiden, besteht darin, Dokumenttypen modular zu definieren und Module mehrfach zu verwenden. In vielen Dokumenttypen, vom Buch über einen Forschungsbericht bis zu einem Artikel, sind Literaturverzeichnisse zu finden. Es empfiehlt sich also, die Struktur eines solchen Verzeichnisses nur einmal zu definieren und die entsprechende Beschreibung dann in die DTD für Bücher, Forschungsberichte und Artikel einzubinden. Um Namenskonflikte zwischen Elementen aus verschiedenen Modulen zu vermeiden, hat das W3C einen Standard zu Namensräumen verabschiedet.

Zur Darstellung der Dokumente sind Formatierungsanweisungen nötig, damit der Browser erkennt, wie Elemente wie z.B. Kapitelüberschriften dargestellt werden sollen. Die extensible Stylesheets Language(XSL) stellt einen allgemeinen Mechanismus für die Darstellung von XML-Dokumenten bereit ([166], S.419).

Im Zuge der XML-Initiative sollen auch die Hypertextmöglichkeiten im World Wide Web verbessert werden. Bislang ist die technische Unterstützung der Hypertext-Idee im Web recht bescheiden. Es bestehen zum Beispiel die folgenden Einschränkungen:

- Verknüpfungen (Links) lassen sich nur in eine Richtung durchführen. Das heißt, dass vom Zieldokument kein Link zurück zeigt, es sei denn, man richtet einen zusätzlichen (unidirektionalen) Link in die Gegenrichtung ein.
- Ein Link kann immer nur auf genau ein Zieldokument zeigen, es sind nicht mehrere optionale Ziele möglich.

In Zukunft sollen die XML Linking Language (*XLink*) und die XML Pointer Language (*XPointer*) das WWW um komplexere Hypertextfähigkeiten erweitern. Dieser Bereich des XML-Umfeldes ist momentan am wenigsten entwickelt. Dennoch bietet der Hypertext ein großes Potential, das durch XLink und XPointer leichter zugänglich wird. Es ist beispielsweise möglich, mit XPointer direkt auf einzelne Elemente eines XML-Dokumentes zuzugreifen. Die Information über die Dokumentenstruktur aus der DTD kann in XPointer genutzt werden.

Schließlich soll ein Aspekt von XML nicht unerwähnt bleiben, der einen großen Anteil am Erfolg von XML hat. Die Möglichkeit, Dokumenttypen formal zu definieren, bedeutet auch, dass Dokumente automatisch (d.h. vom Parser) auf strukturelle Korrektheit überprüft werden können. Zusammen mit den enthaltenen Metadaten (in Form der Elementnamen) bietet XML an, Daten zwischen verschiedenen Anwendungen in einer Netzumgebung auszutauschen. Ein einfaches Beispiel sind Banktransaktionen, etwa Überweisungen. Die einzelnen Bestandteile einer Überweisung lassen sich als XML-Elemente in einer Überweisung-DTD beschreiben [83] (Konto-Nummer, Bankleitzahl, Betrag usw.). Sollen nun die Daten eines Überweisungsvorgangs zwischen zwei Computern ausgetauscht werden, kann der Empfänger allein durch Vergleich mit der DTD verifizieren, ob alle Informationen enthalten sind. Fehlt beispielsweise der Betrag, kann das bereits beim Parsing erkannt werden. Wenn die momentane Tendenz

anhält, wird XML die Grundlage für einen großen Teil des wirtschaftlichen Datenaustauschs (Electronic Data Interchange, EDI) darstellen ([166], S.418).

Zusammenfassend können folgende Argumente für den Einsatz von XML geliefert werden [83]:

- Offenes Format: Der XML-Standard ist offen zugänglich. Im Gegensatz zu proprietären Formaten eines bestimmten Herstellers, sind XML-Daten unabhängig von einem einzelnen Softwareprodukt oder -anbieter.
- Systemunabhängigkeit: XML ist unabhängig von einem Betriebssystem oder von einem Computersystem (Hardware).
- Medienneutralität: XML-Dokumente sind unabhängig von einem Ausgabeformat oder -gerät (Papier, Bildschirm usw.).
- Anpassbarkeit: XML ist kein starres Format, sondern kann und muss auf die jeweiligen Bedürfnisse zugeschnitten werden.

Ein Nachteil ist im hohen Initialaufwand zu sehen. Der Einsatz verlangt üblicherweise nach der Analyse der Dokumentstruktur, dem Entwurf der DTD und der Erstellung einer Layoutbeschreibung. Je nach Umfang der Anwendung ist dabei die Zuhilfenahme von Experten sinnvoll.

XML bietet sich beispielsweise für den Verlag an, der seine fachlich begabten, aber typographisch und gestalterisch weniger versierten Autoren davon abhalten möchte, Hand an das Layout zu legen. Da XML in seiner reinen Form gar keine Formatierungsinformationen enthält, kann der Autor in diesem Punkt keinen Fehler machen. Aus Autorensicht kann man sagen, dass die Arbeit leichter wird, weil sich die Aufbereitung des Textes nach dessen Inhalt und nicht mehr nach dessen Darstellung richtet.

3.1.3.3 Erweiterbarkeit der WWW-Clients-Software

Neben der Spezifikation des Transports, HTTP, und der Spezifikation eines Hypertextformates, HTML/XML, wird das WWW vor allem durch die Plattform definiert, die die inzwischen sehr umfangreichen WWW-Browser darstellen. Diese Browser, z.B. Microsoft Internet Explorer, Netscape Navigator oder Mozilla, zeichnen sich durch einige Erweiterungs- und Integrationsschnittstellen aus.

Java [36]. Die Virtual Machine der Programmiersprache Java von Sun Microsystems implementiert abstrahiert eine wohl-definierte Plattform vom zugrundeliegenden Rechner mit seinem Betriebssystem. Als solche Plattform ist sie eine ideale Ergänzung für WWW-Browser, um Applikations-Klientenlogik mit den Möglichkeiten des WWW zu verbinden. Alle außer den kleineren Embedded-Browsern oder noch unreifen Browser-Projekten bieten die Integrationsmöglichkeit von WWW mit Java.

ECMAScript[6]. Eine Scriptsprache, die in ihren propritären Formen oft noch Javascript genannt wird. Sie wird durch Verweis oder auch direkt textuell in (Hypertext-) Dokumente eingebunden.

MIME-Types [75]. Sie werden benutzt, um neben den obigen Möglichkauten auch externe Anwendungen als Zielapplikation für bestimmte Dokumenttypen festzulegen.

Plugins. Browser-Plugins erlauben es, dass der Browser über eine festgelegte Schnittstelle die Präsentation eines Dokuments an eine andere Applikation abgibt. Die Plugin-Schnittstelle wird zum Beispiel in den aktuellen Mozilla- und Netscapeimplementierungen benutzt, um Java (s.o.) zu integrieren.

3.1.3.4 Multi-Tier-Architekturen

Eine weitere Möglichkeit, um über die Plattform WWW spezifische verteilte Applikationen zu generieren, ist die Integration von extrerner Funktionalkität mit den WWW-Servern (bzw. WWW-Proxies) in so genannten *Multi-Tier-Architekturen*. Hierbei wird das WWW zur Präsentation und zur Verteilung von Präsentationsdaten eingesetzt, während die präsentierten Daten von einer weiteren Anwendungsschicht (englisch: *Tier*) generiert oder allgemein geliefert werden. Modulare Integration von eigenem Code und Kommunikation von Inhaltsdaten, Anfragen und Prozeduralen Aufrufen sind die Grundlagen, um auch die Erstellung komplexer verteilter Applikationen zu vereinfachen.

Integration eigener Module in einen WWW-Server erfolgt über den Aufruf von externen Programmen zur dynamischen Generierung von Präsentationsinhalt (*CGI*). Effizienter und sicherer noch ist die Nutzung von Modulen interpretierten Programmcodes, die in einer virtuellen Maschine ausgeführt werden, die nur einmal gestartet werden muss und dann wiederbenutzt werden kann. Beispiele hierfür sind *Java-Servlets* [16] oder *mod_perl* [8].

Die mit XML gegebene breit akzeptierte Möglichkeit, Übergabeformate von Daten Plattformübergreifend und Standard-konform zu definieren und zu nutzen (sh. Abschnitt 3.1.3.2), wird ergänzt durch die Möglichkeit, prozedurale (RPC [110][111]) oder deklarative (SQL [24], XQuery [149]) Aufrufe von Funktionalität zu versenden.

Diese Mechanismen ermöglichen es zusammen, dass mit Standard-Services (ein wichtiges Beispiel: SQL-Datenbanken) und Standardbausteinen (XML-Werkzeuge, Servlet-Engine etc.) effizient und flexibel verteilte Applikationen gebaut werden können.

Ein noch offenerer Ansatz wird vom W3C, aber auch von Microsoft und anderen verfolgt. Dabei wird ein RPC-System über XML (für das Marshaling), welches wiederum über HTTP übertragen wird, realisiert. Da XML und HTTP von nahezu allen Plattformen verstanden werden kann, wird eine große Plattformunabhängigkeit erreicht.

Entwickelt wird diese Technik unter dem Namen XML-RPC [110][111], WebBroker [80][81] sowie SOAP (Simple Object Access Protocol) [115].

3.1.3.5 Bewertung

Das WWW ist eine hervorragende Plattform für verteilte multimediale Anwendungen, um sehr viele Inhalte für den Zugriff bereitzuhalten und zu integrieren. Besonders nützliche Eigenschaften hier sind die Offenheit der Schnittstellen, Protokolle, Formate, aber auch deren meist zügige Anpassung und Erweiterung im Rahmen des W3C. Viele Elemente des Systems sind sehr einfach zu handhaben im Gegensatz etwa zu CORBA, und durch den Netzwerkeffekt der

sehr weiten Verbreitung kann die Realisierung oder Erweiterung eines WWW-basierten Systems meistens komplett auf vorhandene und vielfach bewährte Lösungen zurückgreifen.

Die Zukunft des WWW wird nicht in einem universell benutzten Protokoll liegen, sondern in der Integration von Protokollen, Formaten und Diensten. Diese Möglichkeit wird bei den Protokollen erst wenig genutzt, etwa bei der Übertragung von Sound-Dateien (Realtime-Plugin). Ferner bietet HTTP/1.1 mit dem Upgrade-General-Header-Feld bereits eine Implementation von Protocol-Switching.

Durch diese umfangreiche Protokollunterstützung würden zwei Dinge erreicht:

1. die Vermeidung der Benutzung des auf Hypertextdokumenten optimierten HTTP als Transportprotokoll [99] und

2. die Vereinfachung von Erweiterungen (diese könnten dann z.B. als Java-Applet implementiert werden und auf ein neues Protokoll verweisen).

Das Ziel, die spezielle Unterstützung für kabellose Kommunikation zu erreichen, wird überflüssig sein, da hier wahrscheinlich Gateways zum Einsatz kommen werden [107].

Von den sehr einfachen, frühen Versionen, die Opfer ihres Erfolges wurden, führte die Entwicklung des HTTP zum aktuellen Standard, dem komplizierten HTTP/1.1. Dieses verursacht zwar deutlich weniger Probleme als seine Vorgänger, doch ist es weit von optimalen Eigenschaften entfernt.

Die Entwicklung des Nachfolgers HTTP-NG wurde nicht als Standard akzeptiert, da zum einen die Architektur des WWW diesen in einem Schritt durch einen anderen ersetzen wollte, und zum anderen, weil es nicht genügend vorhandene Techniken beachtete.

Interessant an HTTP-NG war sein modularer Aufbau, sowie insbesondere die Transportschicht. Das WebMUX-Protokoll aus HTTP-NG wäre ein guter Kandidat für eine Integration mit HTTP/1.1. Eine Erneuerung des WWW und eine Umstellung auf ein verteiltes, objektorientiertes Modell ist jedoch nicht wirklich evolutionär aus den bestehenden WWW-Techniken zu bilden. Die Erfolglosigkeit von HTTP-NG zeigt, dass der Erfolg des WWW den folgenden Zustand zumindest mittelfristig zementiert hat: die Einfachheit, Flexibilität und Erweiterbarkeit werden durch weniger Kontrolle erkauft. Dies umfasst u.a. weniger Kontrolle über den konkreten Einsatz der Ressourcen des Netzwerks und der Betriebssysteme, weniger Kontrolle über die wuchernde Vielfalt der in einer Anwendung integrierten Techniken und keine global verbindlichen Schnittstellen zur Kontrolle der Inhaltssemantik. Wie weiter unten die Diskussion in Abschnitt 3.2.1 zeigen wird, steht der hervorragenden Eignung für Medien-Auslieferung im heterogenem Umfeld eine nur geringe Unterstützung verteilten Authorings gegenüber.

Die Technik WWW entwickelt sich derzeit besonders im Bereich der Codierung strukturierter Information, hier XML.

Tabelle 2: Bewertung des WWW als Grundlage verteilten Multimedia-Authorings

Positiv	Negativ
+ sehr gut geeignet für Hypermedia-Auslieferung	- kein systemweit verwendbarer Mechanismus zum Upload (sh. auch Abschnitt 3.2.1)

Tabelle 2: Bewertung des WWW als Grundlage verteilten Multimedia-Authorings

Positiv	Negativ
+ einfache, flexible Integration verteilten Inhalts	- keine inhärente Unterstützung von Konsistenz und Zuverlässigkeit der Inhalte
+ einfache, flexible Integration von Services und Datenformaten	- kein systemweiter Mechanismus zur Benutzer-Authentisierung und Autorisierung
+ einfach und flexibel einzusetzen	- keine systeminternen low-level Mechanismen (Synchronisation, Priorisierung, Scheduling)

3.1.4 CORBA: Mehr als Middleware

Middleware sind Systeme, die die Integration verschiedener Teile einer Applikation transparent ermöglichen. Dazu gehört vor allem die Abstraktion vom Ort eines Funktionsaufrufs, die Möglichkeit zum *remote procedure call*, oder *RPC*. Middleware kann noch einiges mehr an nützlichen Abstraktionen bieten, vor allem im Zusammenspiel mit objekt-orientierten Strukturansätzen. Hierzu gehören z.B. die Abstraktion von der Plattform durch sprach- und plattformunabhängige Schnittstellen, Zeitunabhängigkeit durch Persistenzdienste und die Benutzung fertiger Strukturen wie etwa Dokumentcontainern zur Manipulation strukturierter Dokumente. Die prominentesten Beispiele für Middleware sind derzeit der ISO-Standard CORBA [45] und das proprietäre System COM [3] (bzw. .NET [104]) von Microsoft.

Durch die Bereitstellung objekt-orientierter Merkmale bietet insbesondere CORBA Möglichkeiten, abstrakte Dienste und "lebendige" Dokumenttypen z.B. für das WWW zu implementieren, was das WWW bisher nicht leistet. Dazu müßte ein einheitliches Ressourcen-Interface, sowie entsprechend in herunterladbarem Code zur Manipulation (etwa ein graphisches Benutzer-Front-End) realisiert werden. Als Protokoll könnte ein Nachfolger von IIOP (Internet Inter ORB Protocol), welches direkt auf TCP/IP anstelle von HTTP aufsetzt, zur Anwendung kommen (z.B: iiop://objectref.operation(parameters)) [103]. Um den entsprechenden Code zu finden, müssten die entsprechenden Localization Middleware Services [45] mit Metadatenansätzen des WWW (etwa RDF [100]) verknüpft werden.

Dieses Szenario macht auf Schwachstellen reiner Middleware-Ansätze aufmerksam. So werden zum einen zwar Mechanismen zu Kontrolle und Signalisierung hervorragend gekapselt zur Verfügung gestellt, Mechanismen zur Präsentation von Kontrollschnittstellen und von Inhalten aber weniger. Die Präsentation zeitkritischer, QoS-orientierter multimedialer Inhalte kann zwar beschrieben und kontrolliert werden, wird aber nicht direkt unterstützt [131]. Wei-

terhin sind die Metadaten zur Lokalisierung von Komponenten viel zu sehr am Code der Komponenten selbst als möglichst explizit an ihrer Applikationssemantik orientiert.

Tabelle 3: Vorhandene Middleware als Grundlage verteilten MM-Authorings

Positiv	Negativ
+ unterstützt gute, modulare, verteilte Applikationsarchitektur	- großer initialer Aufwand
+ generische Dienste (Lokalisation von Komponenten, Persistenz, Sicherheit...)	
+ unterstützt lokale Manipulation von komplexen Dokumenten	- wenig Unterstützung spezifischer Multimedia-Charakteristika (Streaming)
	- Medienspeicherung nicht unterstützt
+ hohe Interoperabilität der CORBA-Systeme	

3.2 Rechtemanagement

Im folgenden werden einige exisitierende Rechtemanagementsysteme betrachtet: zum einen *WebDAV* (Distributed Authoring and Versioning), eine HTTP-Erweiterung, die speziell für den Umgang mit Multimediadokumenten im Kontext des WWW geschaffen wurde, zum anderen bekannte und in der Praxis verwendete dateiorientierte Rechtemanagementsysteme.

3.2.1 WebDAV

3.2.1.1 Allgemeines

Das WWW wurde als Lese- und Schreibmedium geschaffen [147].

DAV steht für Distributed Authoring and Versioning, also für verteilte Erstellung/Bearbeitung sowie Versionierung von Dokumenten aller Art. WebDAV steht somit für DAV im WWW und stellt als solches eine Erweiterung zum WWW-Protokoll HTTP dar.

Da WWW-Ressourcen mit WebDAV direkt auf dem Webserver bearbeitet werden können, kann man sagen, dass WebDAV das WWW zu einem Netzwerkdateisystem erweitert sowie die Grundlage für ein Dokument-Management-System (*DMS*) schafft [143][109].

Den Ausschlag für die Entwicklung von WebDAV gab der Wunsch, das WWW gemeinsam für die Entwicklung von Software nutzen zu können. Das Versenden von Dateien per EMail war zu unpraktisch, fehleranfällig und erfolgte ohne Versionskontrolle, so dass man sich wünschte, die vorhandene Infrastruktur des WWW besser zu nutzen [140].

Um diese Ziele zu erreichen, muss HTTP um eine Schreibmöglichkeit erweitert werden. Der Grund, warum man nicht auf das File Transfer Protocol FTP zurückgreift, ist, dass die

Nutzung von HTTP(/1.1) mehrere Vorteile bietet, wie z.B. starke Authentifizierung, Verschlüsselung, Proxy-Unterstützung, Caching, Persistant Connections sowie Pipelining [143].

Ferner benötigt man eine Möglichkeit konkurrierende Zugriffe zu koordinieren. Dies wird durch das von Datenbanksystemen bekannte Sperren (Locking) von Datensätzen (Objekten/Ressourcen) erreicht. Allerdings wird beim WebDAV-Protokoll das Lost-Update-Problem standardmäßig nur teilweise gelöst, so wird im Kerndokument für die Ressourcen eine Sperrung nur für den Schreib- nicht jedoch nicht für den Lesezugriff vorgesehen [109][141].

Ein weiterer Vorteil von WebDAV ist die Möglichkeit, Daten über die betreffenden Objekte zu halten. Diese sogenannten Metadaten sind Informationen, bzw. Eigenschaften des jeweiligen Objektes und ermöglichen außerdem eine effektive Suche mit Hilfe des noch zu spezifizierenden *DASL* (DAV Searching and Locating) Protokolls. Sie werden als XML-Dokumente realisiert [145].

Eine weitere Funktionalität, die geboten wird, sind die Namespace Operations. Sie realisieren die Möglichkeit, den Server anzuweisen, Ressourcen zu kopieren und zu verschieben, sowie hierarchische Listen, ähnlich einem Verzeichnisbaum abzufragen.

Wie aus dem Namen WebDAV ersichtlich, wird später ebenfalls die Möglichkeit des Versionsmanagement gegeben sein, es können also mehrere Versionen einer Ressource gehalten werden, verschiedenen Personen können gleichzeitig unterschiedliche Versionen einer Ressource erzeugen oder bearbeiten. Dadurch kann später nachvollzogen werden, wer welche Änderungen zu welchem Zeitpunkt vorgenommen hat.

Unter WebDAV besteht die Möglichkeit mehrere Ressourcen (und Verknüpfungen zu Ressourcen) in Collections zusammenzufassen. Diese Collections sind mit Ordnern in herkömmlichen Dateisystemen vergleichbar.

Die für die vorliegende Arbeit wichtigste Funktionalität ist die der Zugriffkontrolle über sogenannte ACLs (Access Control Lists), die die HTTP-Digest-Authentication benutzen. Die Zugriffkontrolle erfolgt hier hierarchisch und kann sehr granular sein. Leider ist ein Standard-Vorschlag für die Zugriffkontrolle über ACLs erst für das Jahr 2000 vorgesehen [144].

Da es sich beim WebDAV-Protokoll, wie bereits geschildert, um eine Erweiterung von HTTP handelt, kommen natürlich neue Methoden und Header hinzu.

Die neuen Methoden nach RfC2518 sind:

- **PROPPATCH:** dient zum Hinzufügen, Ändern und Entfernen von Eigenschaften.
- **PROPFIND:** dient zum Beschaffen der Eigenschaften einer Ressource und/oder deren Mitglieder wenn es sich um eine Collection handelt.
- **COPY:** dient zum Kopieren von Ressourcen inklusive deren Eigenschaften.
- **MOVE:** dient zum Verschieben von Ressourcen inklusive deren Eigenschaften.
- **MKCOL:** dient zum Erstellen von Collections.
- **LOCK:** Sperrt die angegeben Ressource.
- **UNLOCK:** Hebt die Sperrung der angegebene Ressource wieder auf.

  Die neuen Header nach RfC2518 lauten:
- **DAV:** dieser Header muss von allen DAV-fähigen Ressourcen als Antwort auf eine OPTIONS-Methode zurückgegeben werden. Dabei wird zwischen genereller DAV-Fähigkeit und zusätzlich Locking-Unterstützung unterschieden.

- **If:** dient zur Konditionierung von Methoden.
- **Depth:** gibt die Tiefe an, auf die sich die entsprechende Methode auswirkt (“0”: nur die Ressource selbst, “1”: die Ressource und alle Mitglieder, wenn die Ressource eine Collection ist, “infinity”: die Ressource selbst und rekursiv alle Mitglieder).
- **Overwrite:** gibt an, ob das eventuell vorhandene Ziel bei COPY/MOVE überschrieben werden soll (vorhergehendes DELETE).
- **Destination:** das Ziel einer Methode als absolute URI.
- **Lock-Token:** dient zum Sperren, bzw. Freigeben von Ressourcen (sowohl Request- als auch Response-Header).
- **Timeout:** dient zum Verbinden von Sperren mit Timeouts, nach denen die Sperrung automatisch aufgehoben wird.
- **Status-URI:** enthält Informationen über den Status von Methoden.

Bemerkenswert ist, dass die 1996 ins Leben gerufene WebDAV-Initiative der Internet Engineering Task Force (IETF) sehr positiv von der Industrie aufgenommen und noch heute unterstützt wird. So arbeiten Firmen wie Microsoft, Novell und Netscape bereits seit 1996 am WebDAV-Projekt mit. Dadurch verwundert es nicht, dass bereits einige Implementierungen des noch jungen WebDAV-Standards existieren, so im Internet Explorer 5, Office 2000, Windows 2000, sowie im Internet Information Server 5 der Firma Microsoft. Auch der Apache-Webserver verfügt über ein WebDAV-Plugin. Mit den Produkten der Firma Microsoft kann das WebDAV-WWW über die sogenannten Webfolders wie ein “normales” Dateisystem angesprochen werden [109].

Sicher auch aus dieser starken Ausgangsposition heraus liebäugeln die “Erfinder” von WebDAV mit der Vorstellung, ein “Überprotokoll” geschaffen zu haben. Dieses Protokoll könnte Ihrer Meinung nach einige Internet-Protokolle obsolet machen, darunter etwa: FTP (File Transfer Protocol), POP (Post Office Protocol), IMAP (Interactive Mail Access Protocol), NNTP (Network News Transfer Protocol), SMTP (Simple Mail Transfer Protocol). Sogar die Nutzung als “wire protocol” wird auch im lokalen Netz vorgeschlagen [144].

Weitere Vorteile, die WebDAV für sich ins Feld führen kann, sind die große, weltweite Verbreitung des WWW und somit von HTTP sowie die große Anzahl von Experten und Entwicklern, die auf diesem Gebiet tätig sind.

3.2.1.2 Rechtemanagement

Die Zugriffkontrolle bei WebDAV wird, wie bereits erwähnt, über ACLs realisiert.Zunächst müssen einige Begriffe definiert werden:

Eine ACL besteht aus mehreren Access Control Entries (ACE). Jeder ACE gewährt oder verbietet den Zugriff auf eine Ressource oder eine Collection. Ein Principal ist ein Benutzer oder ein Gruppe von Benutzern, denen spezifische Rechte gewährt oder entzogen werden können. Ein Principal ist eindeutig identifizierbar [139].

Einige Ziele, die mittels ACLs erreicht werden sollen, lassen sich am besten anhand von Szenarien darstellen. So soll erreicht werden, dass der Besitzer von mehreren Dokumenten Rechte für jedes Dokument (jede Ressource) individuell vergeben kann. Außerdem muss es

möglich sein, dem Mitglied einer über bestimmte Rechte verfügenden Gruppe eines dieser Rechte zu entziehen.

Weiterhin soll es möglich sein, die Administration von Ressourcen zu delegieren, jedoch ohne es der Person, die die Administrationsaufgabe übernimmt, zu erlauben, auf die Rechte der delegierenden Person Einfluß zu nehmen.

Folgende Rechte sollen in WebDAV implementiert werden:

- Ändern des Inhalts einer Ressource
- Ändern der Eigenschaften einer Ressource
- Löschen einer Ressource
- Hinzufügen eines Kindes zu einer Collection
- Lesen der ACL einer Ressource oder einer Collection
- Ändern der ACL einer Ressource oder einer Collection
- Löschen einer Kindressource einer Collection
- Anzeigen des Inhaltes einer Collection

Es gibt keine Möglichkeit zur Darstellung von Gruppen in HTTP und WebDAV und die Implementierung von Rollen in einem ACL-System ist nur optional, da diese sich sehr schwierig gestaltet. Ferner gibt es keine zeitabhängigen Zugriffrechte.

Der konkrete Protokollvorschlag aus dem Jahr 1997 definiert weiterhin, dass ACLs zu den jeweiligen Ressourcen gehören und, sofern kein expliziter Eintrag vorhanden ist, sämtliche Rechte für den jeweiligen Principal verweigert bleiben. Außerdem werden zusammengesetzte Principals erwähnt - also doch eine Form von Gruppen?

Daraus ergibt sich die grundlegende Regel, dass ein spezielles Recht ein generelles Recht überschreibt. Diese Regel muss man sowohl auf Ressourcen, als auch auf zusammengesetzte Principale anwenden.

Wird eine neue Ressource kreiert, so wird ihr zunächst der jeweilige Besitzer zugeordnet. Die neue Ressource erbt die ACL der Ressource zu deren Inhalt sie gehört. Existiert keine solche Ressource, so hat die neue Ressource eine leere ACL.

Bei der oben beschriebenen Vererbung existieren zwei Varianten, bei der ersten werden Änderungen, die an der ACL übergeordneten Ressource vorgenommen werden, mit der untergeordneten Ressource abgeglichen (dynamisch), bei der zweiten nicht (statisch).

Die Eigenschaften einer Ressource können über eine eigene ACL verfügen, diese muss standardmäßig von ihrer Ressource dynamisch vererbt werden.

Es werden folgende, selbsterklärende Rechte definiert (vgl. auch Anforderungen oben):

- **all**
- **read**
- **write**
- **delete**
- **createchild**
- **deletechild**
- **writeowner**
- **readacl**
- **writeacl**

Weiterhin werden zwei XML-Elemente als Principal definiert:

- **allprincipals**: spezifiziert Rechte die alle Principale haben
- **authprincipals:** spezifiziert Rechte die alle authentifizierten Principale haben.

Die Steuerung der ACLs erfolgt über Methoden, wobei der Request-Body benutzt wird, um die ACL einer Ressource oder deren Eigenschaften zu ändern. Dabei werden vorhandene ACE durch die neuen überschrieben, bei einem leeren Request gibt es keine Änderung.
Der Response enthält die ACL der assoziierten Ressource sowie deren Eigenschaften. Insbesondere wird über den Erfolg des vorhergehenden Requests berichtet. Außerdem werden der Besitzer, aclinheritance und, natürlich, die ACL XML-Elemente übergeben.

XML-Elemente:

- **acl:** spezifiziert eine ACL, jede Ressource muss über eine ACL, wenn auch leer, verfügen.
- **owner:** (Parent: acl) spezifiert den Besitzer einer Ressource.
- **aclinheritance:** (Parent: acl) beschreibt die Vererbungsart, die auf die Kinder der assoziierten Ressource angewandt wird. Der Standardwert ist *dynamic*, andere Werte sind: *static, none, extension.*
- **inheritance:** (Parent: acl) spezifiziert, ob die ACL ihre Werte dynamisch oder gar nicht vererbt.
- **principal:** dient zum Identifizieren eines Principals.
- **ace:** (Parent: acl) enthält Zugriffrechteinformationen, die mit einem oder mehreren Principalen verknüpft sind.
- **grant:** (Parent: ace) identifiziert die Rechte, die dem assoziierten Principal garantiert sind.
- **deny:** (Parent: ace) identifiziert die Rechte, die dem assoziierten Principal entzogen sind.
- **property:** (Parent: acl) realisiert ACLs für Eigenschaften.

3.2.2 UNIX

Im Betriebssystem Unix und seinen Derivaten ist nur eine sehr einfache Rechtemanagementfunktionalität implementiert. Bekannt sind dem System Benutzer, Gruppen und Dateien, wobei die Verzeichnisse eine besondere Form der Dateien darstellen.

Jeder Benutzer ist über eine UID-Nummer (kann vom Ersteller vergeben werden) eindeutig identifizierbar. Außerdem besitzt jeder Benutzer noch eine GID (Group Identification), mit der seine Zugehörigkeit zu einer Gruppe festgelegt ist. Ein Benutzer kann in mehreren Gruppen Mitglied sein, jedoch wird eine Gruppe zu seiner primären Gruppe. Die Gruppenverwaltung in UNIX obliegt einzig dem Benutzer root. Dieser Benutzer besitzt alle Rechte im UNIX-System und ist nicht einschränkbar.

Der Ersteller einer neuen Datei oder eines neuen Verzeichnisses ist der Besitzer, wobei auch gleichzeitig die primäre Gruppe des Erstellers mit dem neu erstellten Element assoziiert wird. Der Besitzer kann nun die Zugriffsrechte auf die Dateien und Verzeichnisse vergeben. Dabei unterscheidet das System drei Klassen. Die erste ist der Benutzer, der mit dem Element assoziiert ist (Besitzer), die zweite die Gruppe, die mit dem Element assoziiert ist und die dritte, dies sind alle anderen, die noch nicht in der Gruppe Mitglied sind. Für jede dieser Klassen kann man die folgenden Zugriffsrechte separat vergeben:

- **read (r)**

- **write (w)**
- **execute (x)**

Diese Rechte sind additiv, das bedeutet, dass eine Person oder Gruppe mehrere besitzen kann und sich die daraus ergebenen Rechte addieren.
Die Rechte lassen sich sowohl auf Dateien als auch Verzeichnisse anwenden, wobei ihre Bedeutung leicht variiert. Auf Dateien angewandt, entsprechen sie sinngemäß ihrem Wortlaut. Um hingegen ein Verzeichnis durchsuchen zu können, benötigt man neben read auch das Recht execute. Wird einer Klasse das Recht zu schreiben (write), auf ein Verzeichnis eingeräumt, so sind die Mitglieder dieser Klasse neben dem Erstellen auch zum Löschen aller Dateien in diesem Verzeichnis befugt. Und dies geschieht unabhängig von den Zugriffsrechten oder dem Besitzstatus der Dateien.

Die Besitzrechte und der Besitzer können bei vielen UNIX-Systemen nur vom Systemverwalter (root) beeinflußt werden. Die mit dem Element assoziierte Gruppe kann vom Besitzer mit dem Befehl *chmod* geändert werden. Es kann eine beliebige Gruppe des UNIX-Systems angegeben werden.

3.2.3 AFS

Das “Andrew Filesystem” AFS ist ein verteiltes Netzwerkdateisystem, welches als physikalischen Speicher den Platz der Festplatten der Computer im Netzwerk benutzt. Dabei wird das zugrundeliegende Netzwerk vom Dateisystem versteckt, so dass das Dateisystem wie eine große Festplatte erscheint. AFS besitzt eine hierarchische Baumstruktur, die Wurzel ist das Verzeichnis /afs. Darunter schließen sich die sogenannten Cells an. Zellen können z.B. eine Firma oder ein Lehrstuhl sein. Unter den Zellen sind die Volumes angesiedelt. Volumes sind Teile von Partitionen von Festplatten (Speicherplatz) und können mit sogenannten Quotas versehen werden. Quotas sind festlegbare Speicherplatzbegrenzungen, die von den Benutzern eines Volumes nicht überschritten werden können. Die Volumes werden über sogenannte Mount-Points in den AFS-Baum eingefügt.

Werden von einem Benutzer Dateien angefordert, so werden sie von dem sogenannten Cache Manager zunächst auf der lokalen Maschine zwischengespeichert und erst beim Schließen der Datei ins AFS-Dateisystem zurückgespeichert. Dabei wird dafür gesorgt, dass das gleichzeitige Editieren durch mehrere Personen koordiniert wird.

ACLs werden in AFS benutzt, um den Zugang zu den Verzeichnissen zu verwalten. Sie werden nur auf Verzeichnisebene angewandt und können nicht auf Dateiebene benutzt werden. Auf Dateien in einem Ordner kann nur zugreifen, wer die entsprechenden Rechte auf den Ordner hat. Über die ACL eines Ordners verfügt einzig der Besitzer des Ordners, damit ist gewährleistet, dass jeder Benutzer über den Zugriff auf sein Home-Directory selbst verfügen kann.

Die Rechte in den ACLs können gewährt oder aber auch für bestimmte Benutzer und/oder Gruppen verboten werden. Die in AFS definierten Rechte lauten:

- **Lookup**
- **Insert**
- **Delete**
- **Administer**

- **Read**
- **Write**
- **Lock**

Sie sind weitestgehend selbsterklärend, Administer erlaubt das Verwalten der ACLs.

Außerdem sind noch acht Zusatzrechte vorgesehen, die jedoch von AFS nicht näher spezifiziert sind. Sie können von Anwendungssoftware für eigene Zwecke benutzt werden ([54], S. 12-4).

Zur Verwaltung des Systems können neben Benutzern auch Gruppen benutzt werden. Drei Systemgruppen werden vorgegeben:

- **system:anyuser**
- **system:authuser**
- **system:administrators**

“system:anyuser” bezeichnet alle Benutzer, die auf irgendeine Art eine Verbindung zu einer Zelle erlangen können. Dazu zählt auch z.B. eine Telnet-Session.

“system:authuser” bezeichnet alle Benutzer, die für die aktuellen Zelle authentifiziert sind.

“system:administrators” bezeichnet die Benutzer einer Zelle, die als Administratoren berechtigt sind, das AFS-System zu verwalten.

Abgesehen von diesen Systemgruppen muss sich, sofern er sie benutzen möchte, jeder Benutzer seine Gruppen selbst definieren. Diese “Protection Groups” werden dann mit *benutzername:gruppenname* bezeichnet und können dann in den ACLs verwendet werden. Der Ersteller einer solchen Gruppe wird automatisch als Besitzer vorgeschlagen und wäre als solcher berechtigt, sie zu administrieren. Dabei kann der Besitzer auch festlegen, ob die Gruppe für andere Benutzer sichtbar ist, damit diese sie für die eigene Administration benutzen können. Ferner ist es möglich, als Besitzer einer Gruppe eine andere einzutragen oder die Gruppe selbst anzugeben. Damit kann die Verwaltung auf mehrere Personen, die jeweiligen Gruppenmitglieder verteilt werden. Der Zugriff auf Informationen über die Gruppen (Mitglieder, Besitzer, etc.) kann sehr vielfältig beschränkt werden.

In den ACLs können nur Benutzer eingetragen werden, die sich an der zugehören Zelle authentifiziert haben. Einzige Ausnahme bildet die Systemgruppe *system:anyuser*, die aber gleich jedem den Zugriff erlaubt. Sie sollte deshalb sehr restriktiv benutzt werden. Für die Praxis bedeutet dies, dass ein Benutzer der zellenübergreifend arbeiten möchte, sich an allen betroffenen Zellen authentifizieren muss.

Zur Verwaltung sind einige Vereinfachungen vorgesehen, so ist es möglich ACLs zu kopieren und gleichzeitig für mehrere Verzeichnisse zu ändern.

3.2.4 Windows 2000

Im folgenden soll das Rechtemanagement des Microsoft Windows 2000-Systems betrachtet werden. Hierbei wird nicht auf die Domänenproblematik und die daraus folgende Unterscheidung zwischen lokalen und globalen Gruppen eingegangen, da diese für die vorliegende Arbeit ohne Bedeutung ist. Berücksichtigt wird ausschließlich das Rechtesystem des NTFS-Dateisystems, weil es die umfangreichsten Rechteverteilungsmöglichkeiten bietet. Es wird also das

Rechtemanagement einer allein stehenden Windows 2000 Workstation oder eines Member-Servers beschrieben.

Das Windows 2000-System benutzt die Konzepte *Benutzer* und *Gruppen* zur Verwaltung der Zugriffsrechte. Dabei existieren bestimmte Systemgruppen, die standardmäßig eingerichtet werden und bestimmte Rechtevorgaben enthalten. Zu diesen Systemgruppen zählen z.B. die Administratoren, die zur vollen Verwaltung des Windows 2000-Systems befugt sind, die Benutzer, die als "gewöhnliche" Benutzer das System nutzen können, und die Hauptbenutzer, die zusätzlich berechtigt sind Benutzer und Gruppen anzulegen und nur diese weiter zu verwalten.

Die Administratoren haben die volle Kontrolle über das System. Jeder Benutzer, der in die Gruppe Administrator aufgenommen wurde, hat künftig alle Rechte und kann nicht weiter eingeschränkt werden. So darf er z.B. alle bereits vorhandenen Benutzer und Gruppen verändern, die Systemgruppen übernehmen. Problematisch ist dieser Ansatz, sobald ein Benutzer über mehrere Rollen verfügen soll. Dann muss er Mitglied mehrerer Gruppen sein und hat damit immer die Rechte, die sich durch die Kombination seiner Gruppenrechte ergeben. Dies kann zu ungewollten Kompetenzanhäufungen und dadurch zu Fehlbedienungen führen.

Das Rechtemanagement auf Dateisystemebene stellt sich wie folgt dar. Jeder Benutzer darf, sofern er die entsprechenden Rechte besitzt, eigene Verzeichnisse und Ordner erstellen. Sobald dies geschehen ist, besitzt er diese Verzeichnisse und Dateien. Eine Ausnahme stellen Benutzer dar, die Mitglied der Gruppe Administratoren sind. Erstellt ein solcher Benutzer eine neue Datei oder ein neues Verzeichnis, so wird die Gruppe der Administratoren als Besitzer eingetragen. Diese verfügt über die Zugriffsrechte auf die Dateien und Verzeichnisse. Folgende Rechte sind definiert:

- **Lesen (R)**
- **Schreiben (W)**
- **Ausführen (X)**
- **Löschen (D)**
- **Berechtigungen ändern (P)**
- **Besitz übernehmen (O)**

Dabei existieren auf Verzeichnisebene mehrere Vereinfachungen durch die Zusammenfassung mehrerer Einzelrechte:

- **Kein Zugriff (Keine)(Keine)**
- **Anzeigen (RX)(Nicht angegeben)**
- **Lesen (RX)(RX)**
- **Hinzufügen (WX)(Nicht angegeben)**
- **Hinzufügen und Lesen (RWX)(RX)**
- **Ändern (RWXD)(RWXD)**
- **Vollzugriff (Alle)(Alle)**

Hierbei gibt die erste Klammer die Einzelrechte auf Verzeichnisebene an, die zweite die Einzelrechte auf Dateien im Verzeichnis.

Auf Dateiebene existieren sinnvoller Weise lediglich die folgenden Zusammenfassungen:

- **Kein Zugriff (Keine)**

- **Lesen (RX)**
- **Ändern (RWXD)**
- **Vollzugriff (Alle)**

Die Rechte sind weitgehend selbsterklärend. “Berechtigungen ändern” (P), gestattet das Verwalten der Datei oder des Verzeichnisses. “Besitz übernehmen” (O) ist eine Besonderheit. Dieses Recht ermöglicht einem Benutzer, den Besitz an einer Datei oder einem Verzeichnis zu übernehmen. Dadurch wird es diesem Benutzer in jedem Fall möglich, auf diese Datei oder dieses Verzeichnis zuzugreifen und es zu verwalten. Das Besondere ist, dass diese Aktion nachverfolgt werden kann, um den Mißbrauch derselben zu vermeiden. Administratoren verfügen immer über dieses Recht, es kann ihnen nicht entzogen werden.

Die Rechte, die einem Benutzer direkt und über Gruppen gewährt werden, addieren sich pro Verzeichnis und Datei, einzige Ausnahme ist, falls dem Benutzer direkt oder über eine Gruppe das Recht “Kein Zugriff” zugewiesen wurde, so sind alle anderen Rechte hinfällig.

Die aktuelle Zugriffsrechteinstellung eines Verzeichnisses vererbt sich auf in diesem Verzeichnis neu erstellten Verzeichnisse und Dateien, kann aber bei diesen dann separat geändert werden. Werden die Rechte für ein Verzeichnis geändert, welches bereits Unterverzeichnisse und Dateien besitzt, so kann getrennt ausgewählt werden, ob diese Rechteänderung für die Unterverzeichnisse und Dateien übernommen werden sollen. Hier wird deutlich, dass diese Vererbung nicht dynamisch erfolgt. Dadurch ist es möglich, dass einem Benutzer oder einer Gruppe in einem Unterverzeichnis andere Rechte zugewiesen werden können, als dem hierarchisch höher liegenden Verzeichnis. Eine Ausnahme besteht, wenn einem Benutzer oder einer Gruppe auf einem hierarchisch höher liegenden Verzeichnis alle Rechte, insbesondere das Leserecht entzogen werden. In diesem Fall kann dieser Benutzer oder diese Gruppe nicht auf Unterverzeichnisse und Dateien zugreifen.

Auf Windows 2000-Server-Systemen ist es ebenfalls möglich, allgemeine Anmeldebeschränkungen zu definieren, so z.B. die Wochenzeiten, in denen einem Benutzer die Anmeldung am System erlaubt wird.

Im Windows 2000-System wird zur internen Identifizierung und Sicherheit ein sogenannter SID (Security Identifier) benutzt, der sämtliche Änderungen an einem Objekt unverändert überdauert. Dieser SID wird bei der Erstellung eines neuen Objektes (insbesondere Benutzer und Gruppe) durch einen Algorithmus generiert, der mit hoher Wahrscheinlichkeit sicherstellt, dass der neue SID einmalig ist.

3.3 Versionierung und Synchronisation der Arbeit vieler Autoren

Versionen sind Ausprägungen von Designobjekten wie z.B. Informationsobjekten [58]. Diese Artefakte sind in Softwareentwicklungsprojekten bestimmte Module der Gesamtsoftware, in DBMS sind Versionen die Ausprägungen von Datenbankfeldern. Diese können beispielweise in Softwareentwicklungsprojekten die einzelnen Programmklassen sein, die je nach Anforderung zu einem Ganzen zusammengefügt werden. In DBMS für beispielsweise CAD Anwendungen sind Versionen die verschiedenen Ausprägungen von Bauteilen.

Dabei können diese Artefakte wiederum eine Gesamtheit von Versionen anderer Artefakte sein und werden Konfigurationen genannt, welche dann je nach Zusammensetzung wieder einzelne Versionen haben. Man denke beispielsweise an eine bestimmte Software, die für verschiedene Betriebssysteme konfiguriert wird und somit verschiedene Versionen hat.

Versionskontrolle bezeichnet die Fähigkeit, Beziehungen zwischen diesen Artefakten verwalten zu können, indem es Konzepte bereitstellt, um diese zu identifizieren und logisch einzuordnen [59]. Die grundlegendste Form der Strukturierung dieser Artefakte ist der Versionsbaum, der die Revisionshistorie der Artefakte wiederspiegelt und somit die einzelnen Phasen der Entwicklung einer Version sichtbar macht.

Die zwei großen Bereiche, nämlich CM Systeme und DBMS für gemeinsame Designprojekte, in denen Versionskontrolle hauptsächlich Anwendung findet, sollen nun näher untersucht werden. Am Ende des Kapitels wird kurz zusammengestellt, welche Relevanz die vorgestellten Methoden für multimediale Applikationen mit vielen Autoren haben.

3.3.1 Grundlagen des Configuration Management

Versionskontrolle bildet die Grundlage für jedes Configuration Management (CM) System. Configuration Management ist dabei die Sammlung von Techniken, die die Konstruktion eines Systems kontrollieren und koordinieren. Da CM Tools auch eine Mehrbenutzerumgebung unterstützen müssen, gehören diese Software Tools zur Klasse der CSCW (Computer Supported Collaborative Working) Systeme. CM Systeme sind insbesondere für komplexe Softwareentwicklungsprojekte zwingend erforderlich, da dadurch erst eine Koordination von Aktivitäten der verschiedenen Benutzer sowie eine konsistente Versionskontrolle möglich wird, um beispielsweise verschiedene Betriebssystemplattformen mit derselben Software unterstützen zu können. Im folgenden erfolgt daher eine Klassifizierung von Software Configuration Management Tools nach unterstützten Prozessen, Modellen und Funktionalitäten, welche in umfassender Weise in [153] dargestellt sind. Da die Versionskontrolle der Komponentenfunktionalität von CM Systemen zuzuordnen ist, wird diese in einem folgenden Abschnitt näher untersucht.

Software Configuration Management Prozesse

Die standardmäßige Definition von CM beinhaltet nach [153] folgende CM Prozesse:

- **Identifikation:** CM Systeme haben die Aufgabe, Produkt-Komponenten zu identifizieren, um diese auf eindeutige Weise abrufen und bearbeiten zu können.
- **Kontrolle:** Unterstützt die Entwicklungskontrolle eines Software-Produktes durch die Entstehungsgeschichte hindurch (Revisionskontrolle).
- **Status Verwaltung:** Berücksichtigt Fertigstellungsstatus von Komponenten, so z.B. Status von Änderungswünschen in Software-Modulen oder Fehlerverfolgung.
- **Audit und Revision:** Ermöglicht das Testen und die Wartung des Produktes in bestimmten Stadien der Entwicklung.

Eine andere Definition umfasst drei Hauptgebiete, die ein modernes CM System unterstützen muss und nach denen diese Systeme auch beurteilt werden können:

- **Konstruktion:** Unterstützt die effiziente Fertigstellung des Produkts bis zur endgültigen Herausgabe durch Unterstützung der Organisation von Entwicklungsprojekten.
- **Prozess Management:** Sichert die prozedurale Abwicklung des Entwicklungsprojektes.
- **Team Arbeit:** CM Systeme sollen die Aktivitäten von mehreren Benutzern und Entwicklern koordinieren und Überschneidungen vermeiden.

Heutzutage stehen gerade für Softwareentwicklungsprojekte viele Tools zur Verfügung, die diese Prozesse automatisieren und damit eine hohe Funktionalität bieten. Diese Eigenschaft erlangen diese Software-Systeme durch Applikation von Modellen, die im nächsten Abschnitt vorgestellt werden.

CM Modelle

CM Tools können auch nach den zugrundeliegenden Modellen unterschieden werden, die sich jedoch nicht ausschliessen. [153] unterscheidet dabei vier gängige Modelle, die CM Tools anwenden.

- **Checkin/Checkout Modell:** Dieses grundlegende Modell verwendet eine zentrale Datenbank (Repository), welche multiple Versionen aller Komponenten eines Produktes enthält. Benutzer können zur Bearbeitung bestimmte Versionen aus dem Repository in den eigenen privaten Bereich kopieren (check out) sowie bearbeitete Versionen einer Komponente wieder hineinkopieren (check in), so dass diese wieder öffentlich werden.
- **Änderungsorientiertes Modell:** Dieses Modell konzentriert sich auf die Änderungen (Deltas) gegenüber einer Ausgangsversion (Baseline) der Komponenten. In diesem Modell sind Versionen die Menge aller Veränderungen, die an der Baseline ausgeführt wurden.
- **Zusammensetzungsorientiertes Modell:** In diesem Modell werden nicht mehr nur Komponenten betrachtet, sondern Systeme von Komponenten. Der Begriff der Versionen von Komponenten dehnt sich auf Versionen von Konfigurationen aus. Die Konsistenz dieser Konfigurationen wird in diesen Modellen thematisiert.
- **“Long Transaction” Modell:** In diesem Modell tritt der Begriff des Arbeitsbereiches (workspace) in den Vordergrund. Hier werden Entwicklungs- und Benutzergruppen voneinander unabhängig gemacht, womit Bearbeitungen der einen Gruppe parallel zu der anderen laufen können. Dieses Modell erlaubt die Verfolgung von Objekten in Ihrer Entwicklung durch verschiedene Arbeitsbereiche.

CM Funktionalitäten

In [4] wird nach unterstützten Funktionalitäten der CM Systeme unterschieden. Hierbei gibt es die Unterscheidung zwischen teamorientierten und prozessorientierten Funktionalitäten.

Die teamorientierten Funktionalitäten berühren eher die technischen Aspekte, wobei die prozessorientierten Funktionalitäten eher die verwaltungs- und koordinierungsorientierten Aspekte der CM Systeme betreffen. Die teamorientierten Funktionalitäten sind nach [153] und [4]:

- **Components:** Identifikation, Klassifikation, Speicherung und Abruf der Komponenten, die das Endprodukt bilden.

- **Structure:** Abbildung der Struktur des Endproduktes.
- **Construction:** Unterstützung bei der Konstruktion des Produktes und seiner Bauteile.
- **Team:** Ermöglichung von Teamarbeit, um ein Produkt zu entwickeln und zu pflegen.

Die prozessorientierten Funktionalitäten werden wie folgt zusammengefasst:
- **Auditing:** Auditierung des Produktes und der zugehörigen Prozesse sowie die regelmäßige Qualitätssicherung.
- **Accounting:** Möglichkeiten zur statistischen Auswertung des Werdegangs eines Produktes, so z.B.: die Entwicklungsmanntage für bestimmte Module einer Software.
- **Controlling:** Kontrolle von Änderungen und Revisionen.
- **Process:** Support der Entwicklungskoordination eines Produktes insbesondere bei grösseren Projekten, die eine straffe prozessuale Vorgehensweise in der Softwareentwicklung erforderlich machen.

Versionskontrolle im eigentlichen Sinne kann somit der Komponentenfunktionalität der CM Systeme zugeordnet werden. Im folgenden werden daher die übrigen Funktionalitäten nicht weiter besprochen. Der nächste Abschnitt beschreibt, was genau unter der Komponentenfunktionalität zu verstehen ist.

Komponentenfunktionalität in CM Systemen durch Versionskontrolle

Die Komponentenfunktionalität von CM Systemen basiert meist auf dem einfachen Checkin/Checkout Modell. Alle möglichen Versionen von Komponenten sowie die unterschiedlichen Zusammensetzungen dieser Komponenten eines Systems sind dabei in einer Repository genannten Datenbank hinterlegt. Grundlage des Aufbaus dieser Repository ist die zugrundeliegende Versionsverwaltung. Daher werden im folgenden die generellen Aspekte der Versionierung beschrieben, da diese erst die Komponentenfunktionalitäten von CM Systemen ermöglicht.

[7] diskutiert die Problematik, dass Versionierung nicht als Konzept sondern als Mechanismus verstanden wird. Daraus resultiert ihrer Meinung nach die Anwendung von Versionierungsmechanismen für Bereiche für die es nicht gedacht ist. Dies sind: Team Kooperationen, History tracing und andere. Dies kann zu Problemen führen, weil je nach Anforderung der Applikation an die Versionskontrolle unterschiedliche Versionierungskonzepte benötigt werden, diese jedoch an die gängigen Mechanismen angepasst werden und nicht der Versionierungsmechanismus geändert wird.

Daher stellen [7] die Versionierung auf eine konzeptionelle Basis und unterscheiden drei Dimensionen der Versionierung.
- **Historical:** Die Versionierung in Abhängigkeit der Entwicklungszeit gibt die einzelnen Revisionen einer bestimmten Komponente wieder. Diese Dimension der Versionierung erlaubt es, die Entstehungsgeschichte von Komponenten zurückzuverfolgen.
- **Logical:** Neben der historischen Versionierung werden Objekte kreiert, die in verschiedenen Varianten gleichzeitig existieren. Sie ist eine Alternative zu einer Revision zu einer bestimmten Zeit.

- **Cooperative:** Hierdurch werden Aktivitäten versioniert, die gleichzeitig (concurrent) an einem Objekt und unabhängig voneinander von verschiedenen Benutzern getätigt werden. Zu einer bestimmten Zeit können gleichzeitige Aktivitäten an einem Objekt mehrere kooperative Versionen haben. Dadurch können mehrere Versionen einer Variante in einem bestimmten Revisionszeitpunkt entstehen. Auch können unterschiedliche Komponenten mit verschiedenen Versionierungshistorien miteinander kombiniert werden. Dazu ist eine zusätzliche Versionierung der Versionsgraphen notwendig.

Die Versionierung von Varianten und Revisionen wird in Versionierungsgraphen repräsentiert, welche Relationen zwischen den einzelnen Varianten und Revisionen eines Objektes repräsentieren [153]. Dabei bedeutet ein Pfeil von Objekt A zu B, dass B aus A entstanden ist (Is-derived-from-Relation). Abbildung 4 illustriert die Abhängigkeiten zwischen den Revisionen und Varianten.

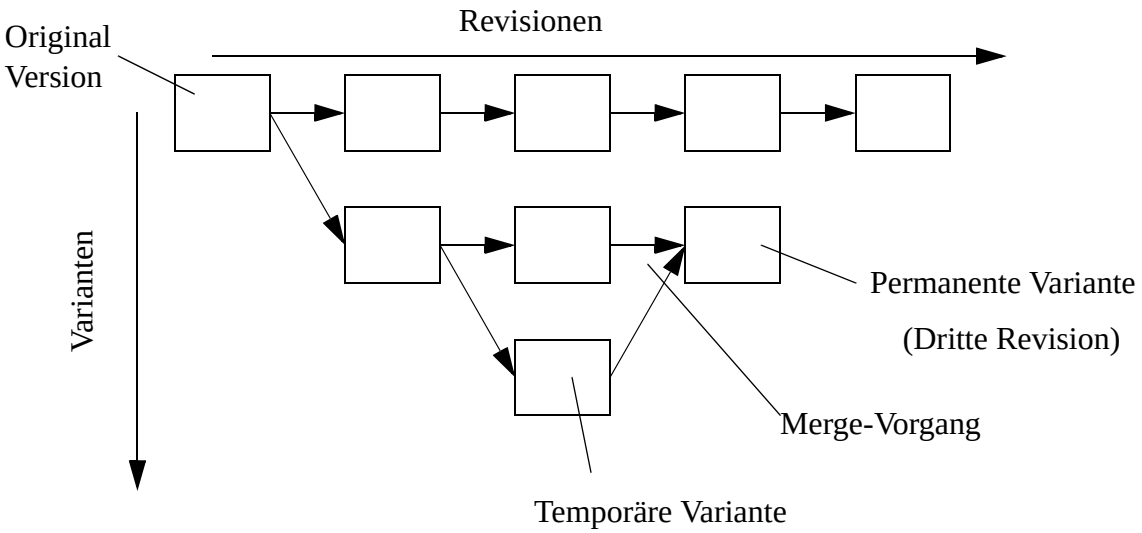

Abbildung 4: Versionierungsrepräsentation in einem Versionengraph

Diese Art von Versionierungsgraphen können nur ein Objekt versionieren. Das bekannteste Tool zur Versionskontrolle mittels dieses Versionsgraphen ist das Revision Control System (RCS) unter dem Betriebssystem UNIX.

Die Revisionen können auch durch das jeweilige Datum der Änderung voneinander unterschieden werden. Die Versionierung von mehreren Objekten und die gleichzeitige Verknüpfung zu Konfigurationen von verschíedenen Objektversionen zu Konfigurationen können diese Versionengraphen alleine nicht leisten. Dazu wurde von [40] das Konzept eines Objekt-Pools vorgestellt, welches die Konfigurationen der einzelnen Versionen und Varianten berücksichtigt.

Somit ist es möglich geworden, komplette Systemversionierungen durch orthogonale Versionskontrolle zu erstellen.

Bei multiplen Konfigurationen ist in diesem dreidimensionalen Objekt-Pool die Verfolgung der Versionen für den einzelnen Benutzer zu komplex. Daher schlägt [40] einen Projektbaum

vor, worauf jedoch nicht näher eingegangen wird, da dies die Selektion von Konfigurationen betrifft.

Configuration Management Systeme im Überblick

Das wohl älteste und mit das einfachste System zur Versionskontrolle ist die mit dem Betriebssystem UNIX mitgelieferte SCCS (Source Code Control System) Kommandosammlung. Da dieses nicht mehr weiterentwickelt wird und auch keine breite Anwendung findet, soll es nicht näher betrachtet werden.

Derzeit sind eine Fülle von verschiedenen CM Tools erhältlich. Mit steigender Verteilung der Entwicklungsarbeit über Kommunikationsnetzwerke steigen auch die Anforderungen an ein CM System. Grundsätzlich wird dabei zwischen frei verfügbaren und kommerziellen Systemen unterschieden. Von den frei verfügbaren Systemen werden die zwei bekanntesten Systeme betrachtet: RCS - Revision Control System und CVS - Concurrent Version System, welche in weiten Teilen der frei verfügbaren CM Software als Grundlage Anwendung fanden.

Auf der anderen Seite wird TeamConnection von IBM als kommerzielles Software Configuration Management Tool betrachtet, welches durchaus höchsten Ansprüchen genügt. Dabei lassen sich vier Anforderungsstufen unterscheiden, welche in aufsteigender Reihenfolge aufgezählt sind:

- **Versionshaltung:** Durch vereinfachte und automatisierte Erstellung von Revisionen und Varianten ermöglicht ein System die Rückverfolgung der Historie. Darüberhinaus ermöglicht die einfache Versionshaltung die Verzweigung zu Parallelentwicklungen und die Wiedervereinigung dieser.
- **Konfigurationen:** Durch Selektionsmechanismen werden die richtigen Versionen zu bestimmten Konfigurationen zusammengebaut. Dies können auch Konfigurationen sein, die gleichzeitig bearbeitet werden. Diese Anforderung ergibt sich bei komplexen Produkten, bei denen nicht mehr nur die einzelnen Komponenten getrennt betrachtet werden.
- **Änderungsverfolgung:** Änderungswünsche und Zustände von Änderungswünschen werden hinterlegt und damit eine Zustandsverfolgung ermöglicht (verworfen, in Arbeit, getestet, freigegeben etc.). So lassen sich auch Zustandsberichte generieren, die eventuelle Verzögerungen in bestimmten Entwicklungsteilen erkennen lassen.
- **Prozessunterstützung:** Unterstützung und Verfolgung der Entwicklung eines Produktes über alle Zyklen (Fertigmeldung, Annahme, Integration&Test, Übernahme). Diese Unterstützung ist insbesondere für sehr große Projekte bei gleichzeitiger Parallelanwendung erforderlich.

Bei der Betrachtung der Software wird zunächst ein grober Überblick über die Funktionalitäten der jeweiligen Systeme gegeben. TeamConnection von IBM unterstützt alle vier Anfordorungsstufen umfassend, wobei sich die Fähigkeiten von RCS und CVS auf einfache Versionshaltung und Konfigurationserstellung und -selektion beschränken.

Da CVS auf RCS aufbaut und diese um bestimmte Funktionalitäten erweitert, wird zunächst RCS beschrieben und darauf aufbauend CVS. Um den Vergleich zu einem typischen kommer-

ziellen Produkt zu schaffen, werden anschliessend die wichtigsten Eigenschaften von Team-Connection dargestellt.

3.3.1.1 RCS - Revision Control System

RCS als System zur Revisionskontrolle wird standardmäßig mit dem Betriebssystem UNIX mitgeliefert. Es ist sehr detailliert dokumentiert (z.B. [53], [151] oder in den Manual-Pages des UNIX-Systems). RCS ist im Prinzip nichts anderes als eine Befehlssammlung unter UNIX zur Versionskontrolle von Dateien. Die Aufgabe hierbei ist das Management von Revisions-Gruppen. Revisionsgruppen sind eine Sammlung von Textdateien, welche als Revisionen voneinander abgeleitet wurden. Das zugrundeliegende Konzept ist hierbei der Revisionsgraph.

Ursprünglich wurde RCS entwickelt, um die Revisionsverwaltung von Programmen zu unterstützen. Es ist aber prinzipiell für Revisionskontrolle von allen Textdateien geeignet, die ständig geändert werden und von denen ältere Versionen aufbewahrt werden müssen. RCS fand auch Anwendung bei der Speicherung von Quelltext für Zeichnungen, VLSI Layouts, Dokumentationen etc.

Somit bietet es die Grundfunktionalitäten an, die ein Versionskontrollsystem haben sollte. Die Kommandostruktur und Einfachheit von RCS prädestiniert es dafür, dass Programme diese Funktionalitäten zwar übernehmen, jedoch um einige Eigenschaften erweitern. So baut auch die Software Concurrent Versions System (CVS) auf RCS auf, welche im nächsten Abschnitt beschrieben ist.

Im einzelnen bietet RCS folgende Funktionaliäten an:

- **Speicherung unterschiedlicher Versionen (Repository, Archiv):**
  RCS legt zu jeder von ihm verwalteten Datei eine Archivdatei an. Dabei wird der jeweiligen Datei die Endung “,v” angehängt. Diese Datei enthält neben dem Ursprungsinhalt noch Zeitmarken, Historyinformationen, Userkennungen usw.. Diese können mit dem Befehl *rlog* abgefragt werden. Hier werden auch die entsprechenden Versionsnummern abgespeichert, wobei RCS keine begriffliche Unterscheidung zwischen Revisionen und Versionen macht. Diese werden durchgängig als Revisionen bezeichnet, was bisweilen zu Verwirrungen führen könnte.
- **Check-In, Check-Out:**
  Basierend auf dem Check-In/Check-Out Modell ist dieses Modell in RCS realisiert. Auf das von RCS verwaltete Archiv darf nur über die RCS-Dienstprogramme zugegriffen werden. Änderungen an den im Archiv abgelegten Dateien können nur vorgenommen werden, wenn diese vorher aus dem Archiv ausgecheckt wurden. Umgekehrt wird eine bearbeitete Datei erst eine neue Revision, wenn sie wieder in das Archiv eingecheckt wird. Beim Chek-In wird bei Bedarf nicht nur eine neue Revisionsnummer vergeben, sondern es werden auch alle anderen Daten entsprechend aktualisiert im Archiv hinterlegt.
- **Revisionskennung:**
  Jede Revision wird durch ein Zahlenpaar gekennzeichnet, die automatisch beim Check-In vergeben wird oder manuell eingestellt werden kann. Die älteste Revision wird mit "1.1" gekennzeichnet. Sie ist die Ursprungsversion. Beim erstmaligen Check-In wird diese mit

den entsprechenden beschreibenden Attributen erzeugt. Bei jeder neuen Revision wird die letzte Zahl um eins inkrementiert.

Es könnnen auch Quadrupel dieser Zahlenkombination als Kennung auftreten. Bei der Kreation von Varianten, wird die Revisionsnummer nochmals um eine zweistellige Nummer ergänzt. Die Revision “1.4” bekommt dann die neue Nummer “1.4.1.1”. Folgende Varianten fangen dann mit “1.4.2.1” an und werden fortlaufend durchnummeriert. Wichtig hierbei ist, dass jede Datei im Archiv eine eigene Nummer hat, die eindeutig ist. Dies wird eben durch Inkrementierung der zweistelligen Zahl und durch Hinzufügen einer Zweier-Zahlenkombination für jede Variante erreicht.

- **Zugriff auf alte Revisionen:**
  Falls nicht explizit eine Revisionsnummer angegeben wird, so wird beim Auschecken der Datei jeweils die aktuelle Revision automatisch ausgecheckt. Um eine bestimmte Revision auszuchecken, wird mit der Option *-r[rev-id]* die jeweils benötigte Revision ausgecheckt.
- **Symbolische Kennung:**
  Mit RCS ist es möglich, verschiedenen Revisionen im Archiv **eine** bestimmte Identifikationsnummer zu geben. Somit wird eine Zuordnung von verschiedenen Dateien und Revisionen zu einer bestimmten Konfiguration geschaffen. Bei Software-Modulen könnte dies die allen gemeinsame Release-ID sein. Somit wird für bestimmte Konfigurationen von Dateien eine endeutige Selektionsmöglichkeit für Konfigurationen ermöglicht. Man sagt, der Entwicklungsstand wird eingefroren.
- **Speicherungsprinzip der Revisionen:**
  Da sich aufeinanderfolgende Revisionen sehr ähneln, wäre es Platzverschwendung, wenn alle Revisionen auf dem Speichermedium gespeichert würden. RCS speichert jeweils die aktuelle Version der Datei und merkt sich die Veränderungen zu den Vorgängerversionen (*diff*-Mechanismus). Dies begrenzt den Platzbedarf für das Archiv auf die Größe der aktuellen Version plus den Informationen, die die Identifikation der Änderungen ermöglichen, also die Änderungen selbst. Dies bedeutet, dass es nur ein File gibt (RCS-File), mit dem die Revisionskontrolle ermöglicht wird.
- **Schutz vor Zugriffskonflikten (Concurrency Control):**
  Beim Check-Out einer Datei kann angegeben werden, ob geplant ist, eine neue Revision anzulegen oder nur lesender Zugriff erfolgt. Im ersten Fall werden automatisch für die ausgecheckte Datei die Rechte *“-r--r--r--”* vergeben. Im letzten Fall jedoch ist es so, dass die Dateirechte auf *“-rw-rw-rw”* gesetzt werden. Hierbei wird jedoch gleichzeitig in der Archivdatei ein Stempel vermerkt, der die aktive Bearbeitung dieser Datei außerhalb des Archivs signalisiert. Dies wird als Locking-Mechanismus bezeichnet. Versucht nun ein anderer Benutzer diese Datei aus dem Archiv auszuchecken, so erfolgt eine Meldung, dass auf diese Datei nicht schreibend zugegriffen werden kann. Diese Funktion könnte jedoch außer Kraft gesetzt werden, falls es aus irgendeinem Grund erwünscht wird, diese Datei zu ändern. Im übrigen sind die Dateien im Archiv lediglich mit einem Check-In-Befehl beschreibbar; die übrige Zeit sind diese Dateien nur lesbar.
- **Kommentierung von Änderungen:**
  Aktuelle Änderungen können beim Check-In einer Datei in das Archiv durch treffende

Umschreibungen kommentiert werden. Dies erleichtert die sinngemäße Wiedererkennung dieser Datei auch durch andere Benutzer und erlaubt, die gemachten Änderungen nachzuvollziehen und mögliche Konsistenzprobleme zu verfolgen. Konsistenzprobleme können gerade bei Softwareentwicklungen entstehen, deren Fehlerrückverfolgung hierdurch sukzessive erfolgen kann.

- **Automatische Kennung:**
  Neben den Kennungen (Bearbeiter, Datum, Änderungskommentar), die im Archiv abgelegt werden, können diese Kennungen auch einfach in die jeweilige Datei integriert werden. Dadurch wird in der Datei selbst kenntlich gemacht, wie die Änderungen zustande kamen.

RCS ist ein System zur Versionskontrolle von Dateien. Jedoch erweist sich schon der Lock-Mechanismus für Concurrency-Probleme als nicht ausreichend. Dieser Lock-Mechanismus kann von einem anderen Benutzer umgangen werden, falls der eigentliche Benutzer die Benutzungsrechte an der Arbeitsversion nicht mit einem Lock versieht. Auf der anderen Seite kann bei einem gesetzten Lock kein anderer Benutzer mehr an der Datei arbeiten. Ein Check-Out dieser gleichzeitig zu bearbeitenden Version würde dem zweiten Benutzer nur Leserechte zugestehen. Nichtsdestotrotz können durch geschickte Kombinationen dieser relativ offenen Funktionalitäten mit RCS ein breites Spektrum an Aufgaben erledigt werden. Nicht umsonst ist RCS auch immer noch Grundlage vieler Configuration Management Systeme. Die Grundidee der Versionskontrolle wurde mit RCS in UNIX integriert. Diese Befehle können in Verbindung miteinander genutzt werden, um auch durchaus komplexe Projekte zu realisieren. Im folgenden werden die RCS Befehle komprimiert dargestellt. Für weitere Details zu den Befehlen sei auf die Manual Pages im UNIX System verwiesen. Die wichtigsten Befehle für die Versionsverwaltung RCS unter UNIX, welche auch von anderen Anwendungen invokiert werden können, sind:

- **co *[optionen]* <file>:**
  Aus dem Archiv wird die spezifizierte Revision herauskopiert und in die entsprechende Arbeitsdatei eingefügt. Wenn nichts anderes festgelegt ist, wird die aktuelle Version herauskopiert.
- **ci *[optionen]* <file>:**
  Dieser Befehl speichert neue Revisionen in die RCS Datei. Falls dieses nicht existiert, wird es neu angelegt. Bei jedem *ci*-Befehl wird nach einem Log-Eintrag gefragt, die die neue Revision beschreiben soll.
- **rcs *[optionen]* <file>:**
  Ändert Dateiattribute oder kreiiert neue RCS-Dateien. Eine RCS-Datei hat Attribute wie z.B.: multiple Revisionen, Access Lists, Change log, Kommentare etc.. Mit der Option *"-i"* wird ein neues RCS-File erstellt und initialisiert. Dabei versucht RCS, die Datei standardmäßig im Unterverzeichnis "/RCS" zu erstellen. Ansonsten wird das RCS-File im momentanen Verzeichnis erstellt.
- **rlog *[optionen]* <file>:**
  Mit diesem Befehl können standardmäßig die Kommentare abgerufen werden, die in der RCS-Datei zu jeder Revision abgelegt sind. Ohne Optionen werden alle Informationen zur Revision dargestellt. Diese enthält: RCS-Pfadname, Arbeitsverzeichnis, Kopfzeile, Aus-

gangszweig im Revisionsgraphen, Acces List zur Datei, Locks, Symbolische Namen, Suffix, Anzahl der Revisionen, Anzahl der Revisionen zur Anzeige, Beschreibender Kommentar. Darauf anschließend erfolgt die Ausgabe der einzelnen Einträge in umgekehrt chronologischer Reihenfolge.

- **rcsdiff *[diff optionen]* <file>:**
  Vergleicht zwei unterschiedliche Revisionen auf Unterschiede in der Textdatei und gibt diese Unterschiede aus. Diese können über die Optionen genau beschrieben werden.
- **rcsfreeze <name>:**
  Dieser Befehl benennt alle aktuellen Revisionen der vorhandenen RCS-Archive mit einer einzigen Markierung. Damit wird indiziert, dass diese Revisionen zu einer gültigen Konfiguration gehören.
- **rcsmerge *[optionen]* <file>:**
  Vereinigt alle Veränderungen, die an einer Datei getätigt wurden in eine zusammengeführte Gesamtrevision. Voraussetzug ist allerdings, dass die Veränderungen in der Datei an verschiedenen Stellen getätigt wurden. Ansonsten wird eine Fehlermeldung generiert, die eine gleichzeitige Veränderung in beiden Dateien in denselben Zeilen anzeigt. Diese Zeilen müssen dann nochmals bearbeitet und abgespeichert werden. Der Befehl *"rcsmerge -r2.8 -r3.4 packetizer.c > packetizer.merge.c"* nimmt die Revisionen 2.8 und 3.4 der Datei "packetizer.c" und fügt diese zur Datei "packetizer.merge.c" zusammen.
- **ident *[-q] [-V]* <file>:**
  Dieser Befehl identifiziert bestimmte Zeichenfolgen, welche mit dem Befehl *co* einem File hinzugefügt wird. Diese Zeichenfolgen haben folgende allgemeine Form: $*Stichwort:Text*$. Mit diesen Zeichenfolgen werden bestimmte Attribute für die Datei bestimmt. Standardmäßig werden von RCS automatisch bei jedem *co*-Befehl folgende Attribute vergeben: Author, Date, Header, ID, Locker, Log, Name, RCSfile, Revision, Source, State. Dabei unterdrückt die Option -q die mögliche Warnung, falls die fraglichen Zeichenfolgen nicht auftreten sollten. Die Option -V gibt die Versionsnummer der Datei wieder.
- **rcsclean *[optionen]* <file>:**
  Vergleicht die aktuelle Datei mit den Revisionen der RCS-Datei. Findet es eine Abweichung, so geschieht nichts. Falls kein Unterschied besteht, so wird die Datei gelöscht. Somit werden nicht mehr benötigte Dateien gelöscht, die zwar ausgecheckt aber nicht weiter bearbeitet wurden.

3.3.1.2 CVS - Concurrent Versions System

CVS erweitert RCS um die Fähigkeit, bei kooperativen Arbeiten Überschneidungen zu organisieren und bei Bedarf zu verhindern. RCS ist somit ein Low-Level System im Gegensatz zu CVS. Die Unterschiede der beiden Systeme sind vergleichbar mit den Unterschieden zwischen einer Maschinensprache und einer höheren Programmiersprache (z.B.: Pascal). Auch bei CVS gibt es ein zentrales Archiv (Repository), in welchem die Revisionen hinterlegt sind. Im Repository werden nicht nur die Änderungen an einer Ursprungsversion gespeichert, sondern es werden alle Versionen dieser Dateien gespeichert. Dies ermöglicht eine verbesserte Funktionalität bezüglich kooperativer Arbeit. Darüberhinaus unterstützt CVS durch die Anbindungsfä-

higkeit an eine Client-Server Architektur die Arbeitskoordination über ein Netzwerk. Die hervorstechendsten Merkmale des CVS sind nachfolgend aufgelistet:

- Gleichzeitige Bearbeitung durch mehrere Benutzer
- Mehrere Entwicklungslinien innerhalb nur einer Repository
- Möglichkeit der Gruppierung von Dateien zu Konfigurationen
- Markierungsmöglichkeit von Dateien mittels sogenannter “Tags”
- Unterschiede zwischen Versionen
- Konfigurierbarer Logging-Support
- Unterstützung von beliebigen Dateien (RCS unterstützt dagegen im Prinzip nur Textdateien)

Im Repository des CVS können mehrere RCS-Files abgespeichert werden. Das Ordnungsprinzip ist hierbei eine Verzeichnisstruktur. Das Repository agiert als logisches Laufwerk für verschiedene Revisionen. An dieses Repository kann auch über ein Netzwerk zugegriffen werden, wofür es spezielle CVS-Clients gibt. Dadurch ist es im Gegensatz zu RCS möglich, multiple Entwicklungspfade für Konfigurationen (in CVS als Module bezeichnet) zu unterstützen.

Eine der Stärken von CVS gegenüber RCS ist es, nicht nur alte Revisionen von Dateien zu verfolgen und abzurufen, sondern diese als sogenanntes “Modul” abrufen zu können, die nichts anderes darstellen als eine Zusammenstellung von Dateien zu einer Konfiguration. Hier wiederum können mehrere Personen an denselben Modulen arbeiten und diese durch einchecken in das Repository als öffentliche Revision zugänglich machen. Diese kooperative Komponente von CVS wird nicht zuletzt durch viele verschiedene Entwicklungen von graphischen Benutzeroberflächen für CVS bestärkt. Diese Oberflächen sind dann meistens die Schnittstelle zu einem CVS-Client, der über ein bestimmtes Netzwerk auf das zentrale Repository zugreift. Einige Beispiele für graphische Benutzeroberflächen für CVS sind: jCVS, WinCVS, CVSweb.

3.3.1.3 TC - TeamConnection

Als kommerzielles Produkt ist TeamConnection auf die Koordination von großen Entwicklungsprojekten ausgerichtet. Es erleichtert und schematisiert die Kommunikation zwischen Entwicklern. Im einzelnen bietet es folgende Merkmale:

- **Configuration Management:** TeamConnection beinhaltet nicht nur die simple Versionskontrolle von Modulen. Darüberhinaus wird über Zuteilung von Status zu den Modulen die Kontrolle des Entwicklungsprozesses erleichtert.
- **Release Management:** Dies bezeichnet die logische Organisation von Objekten, die mit einem Produkt verbunden sind. Releases können gemeinsam entwickelt, getestet, freigegeben und ausgeliefert werden.
- **Version Control:** Versionskontrolle im eigentlichen Sinne. Diese beinhaltet nicht nur die Versionierung von einzelnen Artefakten, sondern auch beispielsweise die Versionierung in Arbeitsbereichen (Long Transaction Modell).
- **Change Control:** TC zeichnet alle Änderungen auf, die an einer Datei oder einem Produkt getätigt wurden. Dies beinhaltet auch z.B. den Status von Änderungswünschen, Bug-fixes etc. Dieser Prozess ist individuell konfigurierbar von strikter bis leichter Kontrolle.

- **Build Support:** Mit TC ist es möglich, die Architektur einer Software vorzustrukturieren. Somit wird die unabhängige Bearbeitung von Teilen der Gesamtarchitektur möglich.
- **Packaging support:** Bei der Auslieferung der Software an den Kunden kann diese individuell zusammengestellt werden.

TC ist für eine Client-Server Netzwerkumgebung ausgerichtet. Es gibt hierbei einen Family-Server, der alle Benutzer- und Entwicklerdaten der TC-Entwicklungsumgebung speichert und bereithält. Diese Daten sind jeweils konsistent in einer Familie. Für ein Projekt gibt es meist mehrere Familien von Entwicklungs- und Benutzerdaten, die jedoch nichts miteinander zu tun haben. Diese können Textobjekte, Binärdateien, Metadatenobjekte usw. sein. Entwickler greifen beim Check-Out ihrer Arbeitsdateien mittels eines TC-Clients auf den Family-Server zu. Die Clients sind in drei Arten realisiert: Als Graphical User Interface (GUI), als Command Line Interface oder als WWW-Client, mit dem auch der Zugriff über das Internet weltweit möglich ist.

Die Zugriffsteuerung der Entwickler auf die Datenbank erfolgt über die Vergabe von User-ID Nummern. Dabei gibt es je Family einen Superuser, welcher erweiterte Rechte hat, die die Koordination der Gesamtarbeit betreffen. Jener Superuser kann einen Entwickler beispielsweise autorisieren, bestimmte Bearbeitungen an Dateien vorzunehmen oder solche Änderungen anfordern.

User-ID können auch mit einer Host-List assoziiert werden, von denen aus der jeweilige User Zugriff auf den Server bekommt.

Revisionen in TC werden Parts genannt. Parts können dabei beliebige Ausprägungen der bearbeiteten Objekte sein. Diese können auch beliebige Kombinationen von diesen Objekten sein.

Innerhalb einer Familie sind die Entwicklungsdaten in Gruppen (*Components*) organiziert. Diese Components sind wiederum hierarchisch mit einer Root als Mutter aller Components aufgebaut. Entlang dieser hierarchischen Ordnung wird auch Zugriffkontrolle auf Entwicklungsdaten geschaffen. Eine Änderungsanforderung kann somit beispielsweise nur von einer Stelle kommen, die dafür auch die Berechtigung hat. Diese Änderungswünsche gehen meistens an Mitglieder von Components, die mindestens eine Hierarchieebene tiefer liegen.

Jedem Entwickler ist ein bestimmter Arbeitsbereich (Work-area) zugeteilt. Hierher kann der Entwickler seine Arbeitspakete aus dem Family-Server herauschecken und bearbeiten. Mehrere Arbeitsbereiche sind in sogenannten Drivers zusammengefasst, um verfolgen zu können, welche Änderungen an einem Objekt, wann, durch welchen Arbeitsbereich getätigt wurden (Long Transaction Modell).

3.3.2 Ausgewählte Verfahren zur Versionskontrolle in DBMS

In einer Vielzahl von Datenbankanwendungen ist inhaltliche Versionskontrolle nicht von großer Relevanz. Dort kommt es vielmehr auf die Schnelligkeit und Lastenverteilung an, beispielsweise in elektronischen Archiven von Kundendaten in Unternehmen. Versionen von Designobjekten werden vornehmlich für Entwicklungsprojekte benötigt, damit die Entwick-

lungsgeschichte zurückverfolgt werden kann. Zusätzliche Funktionen müssen vom DBMS unterstützt werden, falls mehrere Anwender an einem Entwicklungsprojekt arbeiten.

Versionierungssyteme für DBMS berücksichtigen die Verknüpfungen der Designobjekte untereinander. Die grundlegende Verknüpfungsart zwischen Objekten ist die Enthaltungsrelation, die anzeigt, dass ein Objekt in einem anderen Objekt enthalten ist. Dies ist beispielsweise der Fall in CAD-Systemen für die Entwicklung von Integrierten Schaltungen, wo bei einer Änderung eines Einzelteiles auch das Bauteil verändert wird, in welchem dieses Einzelteil zur Anwendung kommt. Analoges kann auch über Dokumente in Dokumentenmanagementsystemen gesagt werden. Falls sich bestimmte Abschnitte in einem Text verändern, verändert sich auch das gesamte Dokument, da dieser Abschnitt darin enthalten ist. Somit kann anhand des Versionierungssystems die Entwicklungsgeschichte des Dokumentes zurückverfolgt werden.

In [25] ist ein Versionsmodell für CAD-Systeme beschrieben. Gleichzeitig wird dort auch ein Überblick über andere Systeme gegeben. Auch in Anlehnung an [25] soll im folgenden mit graphischen Erläuterungen gearbeitet werden, um das dort beschriebene Versionierungsmodell vorzustellen.

Die Daten in dem Modell sind zunächst hierarchisch verknüpft, d.h. Objekte können in anderen Objekten enthalten sein bzw. enthalten selbst wiederum andere Objekte. Darüberhinaus gibt es noch Objekte, die als äquivalent gelten, jedoch getrennt abgelegt werden müssen. Ein Beispiel hierfür ist die Arithmetisch Logische Einheit (ALU) eines elektronischen Bauteiles. Dieses kann eine Layout-, eine Transistorverknüpfungs- und eine Funktionsbeschreibungsansicht haben. Diese drei Ansichten sind mit einer Äquivalenzbeziehung miteinander verknüpft. Da ein Bauteil aus verschiedenen Komponenten besteht und diese wiederum verschiedene Versionen haben, besteht eine Version eines Bauteils immer aus einer bestimmten Kombination von Versionen anderer Objekte. Diese werden Konfigurationen genannt. Das Datenmodell ist zusammengefasst in der Abbildung 5.

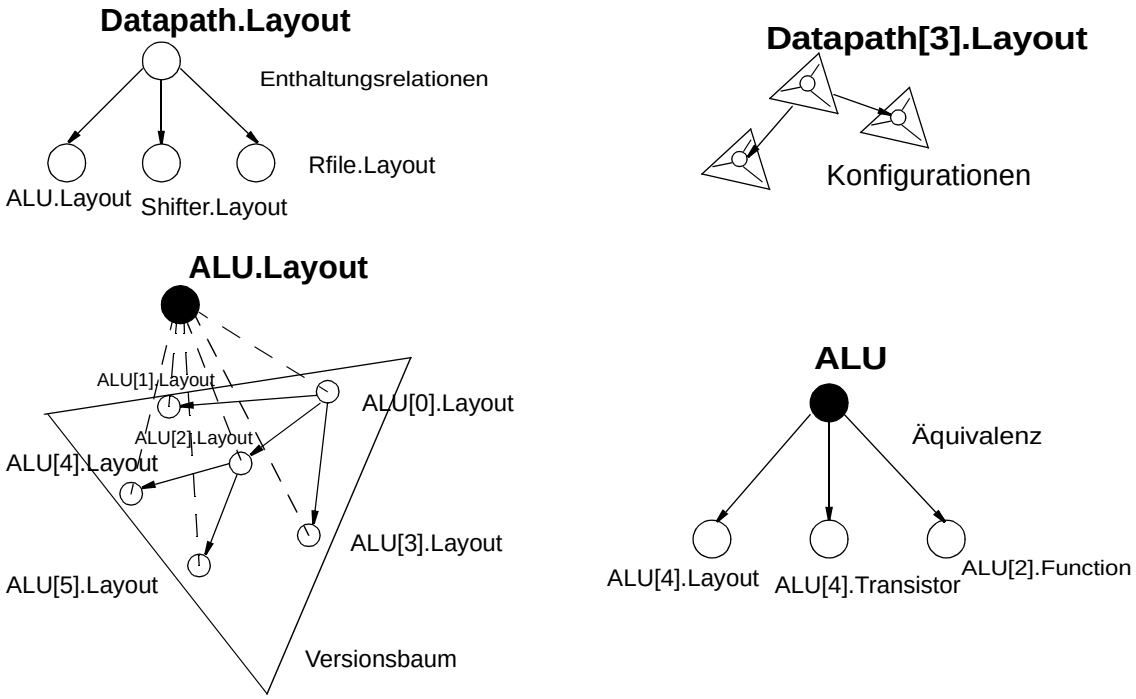

Abbildung 5: Datenmodell eines CAD-DBMS nach [25]

Die Objekte sind nach folgendem Schema benannt: "Name[Versionsnummer].Typ". Der Versionsbaum wurde schon in Kapitel beschrieben.

In der Datenbank gibt es Operationen, welche die gegebene Datenstruktur verändern. Dabei geht es um folgende Problembereiche, die angesprochen werden:

- **Currency:** Es ist oft das Erfordernis vorhanden, immer die aktuelle Version einer Einheit zu bestimmen. Dies kann beispielsweise die Einheit mit dem letzten Erstelldatum sein oder die aktuelle Version kann explizit bezeichnet werden.
- **Dynamische Konfigurationen:** Dynamische Konfigurationen entstehen durch Verweise von Objekten auf andere Objekte. Diese Verweise werden in einem nächsten Schritt in Enthaltungsrelationen umgewandelt.
- **Workspaces:** Workspaces sind bestimmte Bereiche, wohin Objekte eingelesen werden. Diese können gemeinsam, privat oder durch eine bestimmte Gruppe genutzt werden. Der Unterschied zwischen den Arbeitsbereichen sind die unterschiedlichen Zugriffrechte auf Objekte.
- **Logische und physische Repräsentation:** Die logische Verknüpfung der Objekte muss auf eine physische Speicherstruktur übertragen werden, die in der Regel nicht hierarchisch, sondern flach organisiert ist.
- **Change/Constraint Propagation:** Change Propagation bezeichnet den Prozess, der neue Versionen automatisch in Konfigurationen einbindet. Constraint Propagation bezeichnet die Behandlung von Änderungen in anderen Ordnungsmerkmalen der Daten. In [25] ist dies die Behandlung der Äquivalenzbeziehungen.

- **Vererbung:** Objekte sind Instanzen von Klassen. Jedoch können Klassen andere Klassen enthalten. Ein Beispiel ist die Äquivalenzbeziehung. Ein Typ kann Teil einer Klasse von Typen sein, welche wiederum andere Typen enthält.

Die Lösung dieser Problembereiche in [25] wird im folgenden beschrieben.

Currency Control wird dadurch erreicht, dass im Versionsbaum eine bestimmte Version eines Objektes als "current" gekennzeichnet wird. Der Pfad im Versionsbaum bis zum Wurzelknoten ist der Hauptpfad. Außerhalb dieses Hauptpfades dürfen keine Versionsnachfolger eingefügt werden. Die Nachfolger dieser "current" Version dürfen jedoch Nachfolger haben.

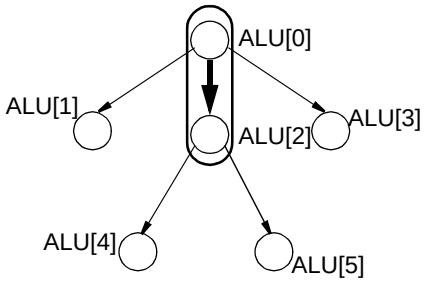

Abbildung 6: Currency Mechanismus in [25]

Mit diesem Mechanismus ist es möglich, nach verschiedenen Kriterien eine Version und dessen Pfad bis zur Wurzel als "current" einzustellen. Beispielsweise kann dieser Pfad von Monat zu Monat, von Nutzer zu Nutzer verschieden sein. Für dieses Kriterium ist dieser Pfad dann die maßgebliche Revisionshistorie des Objektes, an welchen weitere Versionen ergänzt werden können.

Bei der Zusammenstellung von **dynamischen Konfigurationen** wird in [25] ein Schichtenmodell angewendet, das Objekte mit gleichen Versionsnummern in Gruppen sammelt. Die Objekte sind in diesem Schichtenmodell über verschiedene Ebenen der Schichten miteinander verknüpft. Durch dieses Modell ist es möglich, eine Art Pfadstruktur für Objektversionen zu bilden, die in einem anderen Objekt enthalten sind. Darüberhinaus wird mit diesen Gruppen auch die entsprechende Abfragereihenfolge für diese Objekte festgelegt.

Bei den **Workspaces** wird ebenfalls ein Schichtenmodell angewendet. Es gibt drei Schichten: Archive, Group und Private. Wenn ein Nutzer eine Version eines Objektes ausliest, wird diese in den Private-Bereich kopiert. Der Nutzer kann diese Version nun verändern. Beim Einspeichern der veränderten Version gibt es für den Nutzer die Möglichkeit, diese in seiner Gruppe zunächst temporär zu speichern. Speichert er es im Archiv, so ist diese neue Version für jeden zugänglich und ist auch gleichzeitig ein Nachfolger der vorher ausgecheckten Version.

Es gibt zwar immer nur ein zentrales öffentliches Archiv für die Daten, jedoch gibt es in den meisten Fällen mehrere Gruppen und mehrere Einzelnutzer.

Bei der Zuordnung von **physischem Plattenspeicher** zur logischen Struktur wird eine Eins-zu Eins-Zuordnung von Plattenspeicher für jedes Objekt vorgenommen. Die Implementation kann daraufhin bei der Abfrage von Objekten nach dem entsprechenden Plattenspeicher schauen und die zugehörigen Daten auslesen.

Change Propagation sorgt dafür, dass beim Erstellen von neuen Versionen die Datenbank derart aktualisiert wird, dass nicht nur die neue Version des Objektes gespeichert wird, sondern dass Objekte, die mit diesem Objekt verknüpft sind, ebenfalls aktualisiert werden. Dabei tritt insbesondere das Problem auf, dass Objekte in mehreren Konfigurationen enthalten sein können. Zur Lösung dieses Problems gibt es nach [25] mehrere Möglichkeiten:

- Die Änderungen an den enthaltenden Konfigurationen werden durchgeführt. Dies ist jedoch zu restriktiv.
- Für alle Konfigurationen, in denen das veränderte Objekt enthalten war, wird eine neue Version erstellt.
- Change Propagation wird in richtung der Wurzel bis zu dem Punkt hinauf durchgeführt, bis zu dem die Enthaltungsrelation eindeutig ist. Ab dem Punkt, wo ein Objekt in mehreren Konfigurationen enthalten ist, wird keine Change Propagation mehr durchgeführt.
- Angabe der Konfiguration durch den Nutzer, die eine neue Version bekommen soll. Dies wird dadurch möglich, dass beim Auslesen eines Objektes nicht nur dieses ausgelesen werden kann, sondern auch die Konfiguration, in denen es enthalten ist und welche dann beim Aktualisieren einen Versionsnachfolger bekommt.

Die **Versionierung** der Klassen erfolgt analog zur Versionierung der Objekte. Diese werden ebenfalls in einem Versionsbaum gespeichert und enthalten Vorgänger-Nachfolgerbeziehungen. Somit sind beispielsweise für bestimmte Objekte verschiedene Typversionen denkbar, die wiederum als Instanzen verschiedene Objektversionen enthalten.

Der Update-Mechanismus nach [25] ist in der folgenden Abbildung 7 zusammengefasst. Dabei wird Objekt B[0] aktualisiert. Hierbei erfährt die Äquivalenzbeziehung eine Änderung und das Objekt, welches B[0] enthält, in diesem Fall A[0]. Die Quadrate repräsentieren den physikalischen Ort, an dem diese Objekte abgespeichert sind, die Kreise die entsprechenden Objekte, die ausgefüllten Kreise, die als Attribut in den Objekten gespeichert sind.

In Abbildung 7(a) ist nach der logischen Sicht die Ortsangaben für die Objekte dargestellt. In Abbildung 7(b) wird eine neue Version von B[0] nämlich B[1] erstellt. Diese Änderung wird an A[0] weiter propagiert, so dass auch eine neue Version von A[0] nämlich A[1] vorhanden ist, die nun B[1] enthält. Die anderen Enthaltungsrelationen von A[0] bleiben für A[1] erhalten, auch die physikalische Adresse ändert sich nicht. Aufgrund der Äquivalenz zwischen B[0] und E[0] wird E[0] genauso wie B[0] aktualisiert. Das logische Ergebnis ist in der Abbildung 7 unten rechts zu sehen.

Ein ähnliches Modell der Versionierung ist in [26] diskutiert. Hierbei wird neben einer Datenstruktur für versionierbare Objekte auch eine Implementierungsarchitektur vorgestellt, die als Plug-in für Datenspeicher konzipiert ist. Dieses Modell speichert statt den kompletten Versionen nur die Differenzen zu den Vorgängerversionen. Beim Auslesen dieser Version wird aus

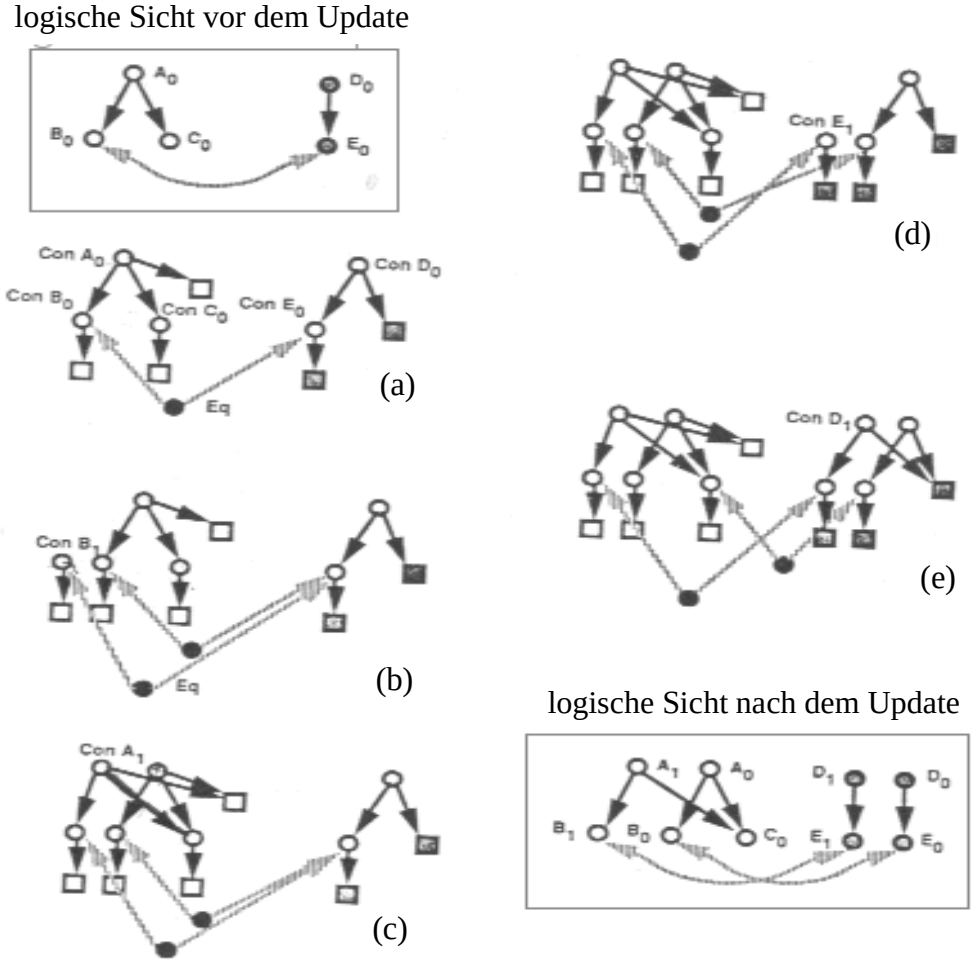

Abbildung 7: Change Propagation nach [25]

dieser Differenz zur Vorgängerversion die aktuell auszulesende Version gebildet und ausgegeben. Wurde in [25] ein Modell zur Versionskontrolle für verknüpfte Objekte vorgestellt und unter anderem Change Propagation als Problembereich gezeigt, beschränkt sich hier die Beschreibung auf die Architektur der Implementation und die Erweiterungen der Datenstruktur von einem Objekt. Dieses Objekt wird dann auf einem Speicherplatz mittels dieses Systems verwaltet.

Ein generisches Objekt zeigt dabei auf den zugehörigen Versionengraph. In den Attributen des Objektes sind Metadaten zu Versionsnummer, nächster Versionsnummer und Default-Version abgelegt. Die Objekte im Versionsgraphen enthalten folgende Metadaten:

- Version Number: Bezeichnet die Versionsnummer.
- OID of version: ID-Kennung der jeweiligen Version. Eine Version kann auch über die ID des generischen Objektes mit Angabe der Versionsnummer ausgelesen werden.
- State of Version: Gibt den Status der Version an. Diese kann sein:
  - Transient: Gibt an, dass diese Version verändert oder gelöscht werden darf.
  - Working version: Diese Version darf nicht mehr verändert, nur noch gelöscht werden.

    - Frozen Version: Dies ist eine stabile Version und kann weder bearbeitet noch gelöscht werden.
    - Deleted Version: Zeigt an, dass diese Version gelöscht wurde. Dies ist wichtig, da spätere Versionen aus einer Differenzbildung zur Vorgängerversion gebildet werden. Somit wird nicht die Differenz von dieser Version, sondern vom Vorgänger dieser Version gebildet.
- R/L/E indicator: Diese Datum gibt an, ob es sich um eine reguläre (R) Version handelt, ob es sich um eine große Version (L) handelt, dessen Differenz zum Vorgänger abgespeichert wurde oder ob es sich um eine große Version (E) handelt, die ohne Differenzbildung zum Vorgänger gespeichert wurde.
- Pointer to parent Version: Gibt die Vorgängerversion an.
- Pointer to child Versions: Gibt die Nachfolgerversionen an.

Neben den in [25] und [26] beschriebenen Modellen zur Versionskontrolle für Objekte ist desweiteren in [37] ein System zur Versionierung von strukturierten Dokumenten vorgestellt. Hier wird ein Modell vorgestellt, welches strukturierte Dokumente betrachtet, mehrere Nutzer annimmt und für dieses ein Versionierungsmodell erstellt. Dieses Modell sei im folgenden vorgestellt.

Dabei geht [37] von einem hierarchischen Dokumentenmodell aus. Ein Dokument enthält Text, Graphiken, Dokumentattribute, Verweise auf andere Dokumente sowie wiederum andere Dokumente. Bei Referenzen wird zwischen dynamischen und statischen Referenzen unterschieden. Statische Referenzen verweisen immer auf bestimmte Versionen von Dokumenten, dynamische Referenzen verweisen auf die gerade aktuelle Version eines Dokumentes.

Die aktuelle Dokumentversion ist immer die zuletzt erstellte Version, da Varianten im Revisionsgraphen nicht zugelassen werden. Der Revisionsgraph besteht hier nur aus einem Pfad ohne Nebenäste, daher ist die letzte Version eines Dokumentes immer die aktuelle.

Beim Erstellen von neuen Versionen eines Dokumentes müssen Notifikationen durchgeführt und einzelne Änderungen propagiert werden. Notifikation findet auf zwei Arten statt: Flag- und Message-basiert. Dabei werden die Nutzer eines Dokumentes bei Änderungen benachrichtigt. Hierzu werden zunächst die Dokumente, für die eine Notifikation gewünscht ist, mit einem Attribut gekennzeichnet, so dass beim Aufruf der Nutzer benachrichtigt werden kann. Nutzer von Dokumenten mit dynamischen Referenzen werden in [37] immer benachrichtigt. Die Notifikation findet bei Nutzern von Dokumenten mit statischen Referenzen nur dann statt, falls eine neue aktuelle Version vorhanden ist oder die derzeit genutzte Version gelöscht wurde. Die Propagation der Änderungen erfolgt wie auch in [25]. Bei dynamisch erzeugten Dokumenten werden bei einer Änderung eines Teils des Dokumentes auch die diese Teile enthaltenden Dokumente aktualisiert und eine neue Version erstellt, die nun die aktualisierte Version dieses Dokumentfragments enthält.

Der grundlegende Unterschied des in [37] vorgestellten Dokumentenversionierungssystems zu den in [25] und [26] ist der, dass der Versionsbaum nur aus einem Pfad besteht. Dies ist zwar eine Einschränkung, jedoch ist die aktuelle Version eines Dokumentes immer eindeutig. In [25] wurde im Versionsbaum nach bestimmten Kriterien (z. B.: Nutzer) immer eine Version als

aktuelle definiert, womit der Pfad bis zur Ausgangsversion der Revisionsgraph für diese Version wurde. In [26] wurde für ein generisches Objekt, welches auf eine Version im Versionsgraphen zeigt, immer eine Version als aktuell definiert, so dass auch eine global eindeutige aktuelle Version vorhanden war, jedoch enthielt der Revisionsgraph auch hier Seitenäste.

In [60] werden neben den schon oben diskutierten Versionierungmodellen drei grundlegende Formen der gemeinsamen Dokumentenbearbeitung vorgestellt: sequentielle, parallele und wechselseitige.

- **Sequentiell:** Bei der sequentiellen Bearbeitung wird die jeweils überarbeitete Version eines Dokumentes an den nächsten Bearbeiter weitergereicht.
- **Wechselseitig:** Bei der wechselseitigen Bearbeitung arbeiten mehrere Benutzer an einer Version eines Dokumentes und realisieren einen Versionsnachfolger.
- **Parallel:** Bei der parallelen Bearbeitung gibt es jeweils für jeden Benutzer eine Version eines Dokumentes, welche dann in einem dritten Vorgang zu einem Dokument zusammengefasst werden.

Diese drei Szenarien veranschaulicht Abbildung 8.

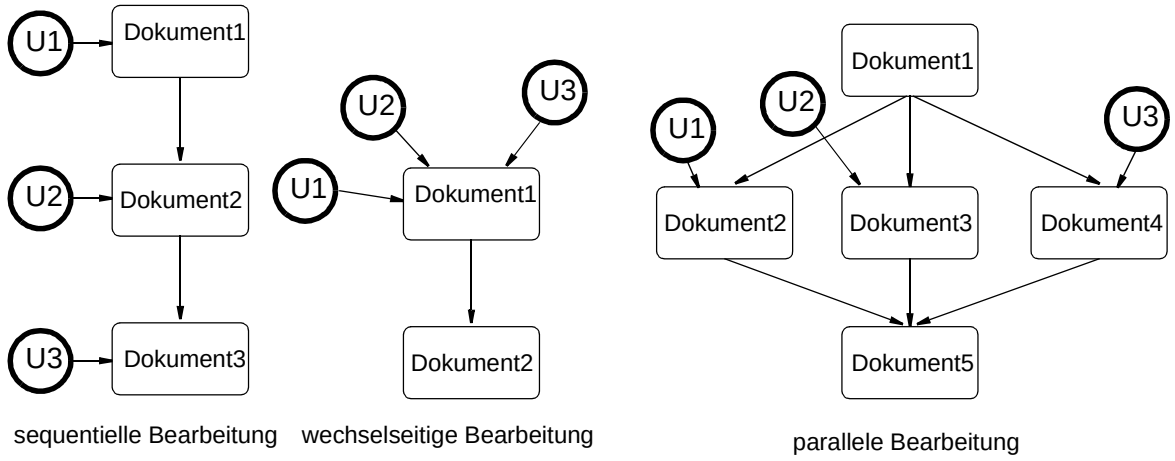

Abbildung 8: Entwicklungsstufen eines Dokumentes

Für die Umsetzung eines multimedialen verteilten Systems ist zu beachten, dass mehrere Benutzer dieselben Präsentationen nutzen werden, jedoch die Erstellung multimedialer Inhalte von dedizierten Systemen "außerhalb" eines Gesamtsystems erfolgen wird. Dies ergibt sich zum einen aus der unstrukturierten Form (BLOB) eines speziellen Mediums, zum anderen aus der Forderung nach Flexibilität: mit festgelegten Formaten als Übergabeschnittstelle (etwa MPEG-1 oder MPEG-2) sollte immer das Werkzeug der Wahl zur jeweils beabsichtigten Manipulation benutzt werden können. Jeder Nutzer wird zunächst unabhängig von anderen Aktualisierungen an Inhalten vornehmen, welche dann nach Freigabe von anderen Nutzern benutzt werden dürfen, d.h. es ist nicht das Bedürfnis vorhanden, zwei von unterschiedlichen Bearbeitern freigegebene Präsentationen zu einem Ganzen zusammenzulegen. Somit bietet sich die parallele Bearbeitung von verschiedenen Versionen der Ressourcen an, aber eine Ver-

einigung von Änderungen in einem "atomaren" Dokument, einem Monomedium, ist in diesem Rahmen nicht vorgesehen (s. Abschnitt 3.3.3).

3.3.3 CSCW und Workflow

MIt Konzentration auf die notwendige Struktur und Semantik von Inhalten einer multimedialen Applikation mit vielen Autoren werden zwei verwandte Aspekte in dieser Arbeit nicht betrachtet.

Das ist zum einen die koordinierte Bearbeitung eines Monomediums durch mehrere Autoren gleichzeitig. Die Probleme hier etwa bezüglich der Synchronisation sind denen bei der Versionierung verwandt, können aber separiert von der Spezifikation der gewünschten Struktur und Semantik der Applikationsinhalte betrachtet werden, indem diese Arbeit Monomedien als die Atome des multimedialen Inhalts verwendet. Soll diese Granularität verfeinert werden, müssen Methoden des CSCW [17] zur Behandlung von Konflikten innerhalb von Monomedien zur Anwendung kommen. Da CSCW Aspekte der nicht-idealen Verteilung addressiert, würden hier Methoden wie Replikation und Konfliktauflösung zum Einsatz kommen. Eine Integration der Ergebnisse unter anderem dieser Arbeit mit solchen Methoden präsentiert On in [38].

Auch dynamische Aspekte des Workflow Management [57] werden hier nicht betrachtet. Da entsprechende statische Aspekte, insbesondere Nutzerinformation, Rollen, Besitz- und Rechteinformationen (wenn auch in anderem Zusammenhang), betrachtet werden, und diese Arbeit die Erweiterbarkeit von Struktur und Semantik der vorgestellten Konzepte explizit vorsieht, sollen die Organisation von Arbeitsprozessen und andere Workflow-Funktionalitäten hier nicht betrachtet werden.

3.4 Schlussfolgerung

Um durch Strukturierung die Komplexität einer verteilten multimedialen Anwendung zu beherrschen, eignet sich der Einsatz von Middleware sehr gut. Die vorhandenen Middleware-Ansätze aber unterstützen Applikationsspezifika besonders in zwei Punkten zu wenig:

1. die Behandlung von Video- bzw. Audio-Daten wird bisher nur in experimentellen Middleware-Systemen direkt unterstützt. Middleware unterstützt die Kontrolle und die Signalisierung der Kontrolle, jedoch muss die Speicherung und der Transport solcher Daten jeweils über andere Mechanismen gelöst werden.
2. die Abbildung der Applikationssemantik, insbesondere der Daten über Autoren und ihre Organisation. Daten über multimediale Inhalte und ihre Zusammenhänge und Daten über den Bezug zwischen Autoren und Inhalten sind komplex strukturiert und kommen in großen Mengen vor. Diese Daten sollten möglichst explizit und in effizienten, dedizierten System vorliegen.

Diese beiden Aufgaben werden viel besser von spezialisierten Datenbanken übernommen als durch die Benutzung von Middleware-Services gelöst. Die Benutzung von Middleware zur Erstellung von verteilten Applikationen mit Datenbanken als Backend zur Bereitstellung von strukturierten, multimedialen und Massendaten weist aber immer noch Probleme hinsichtlich der Präsentation auf: zum einen die Präsentation der Strukturen der Inhalte und der Anwen-

dung an sich, zum andern die Präsentation von Audio- und Videodaten (allgemeiner: die Präsentation mit Quality of Service).

Der zweite Punkt, multimediale Präsentation mit QoS, wird hervorragend von MPEG addressiert. MPEG allein reicht als Framework für eine multimediale, verteilte Anwendung mit vielen Autoren nicht aus, vor allem, da der Fokus von MPEG auf der qualitativ hochwertigen Präsentation von mulitmedialen Daten liegt und es die Darstellung und den Transport von Applikationssemantik wenig unterstützt. Für die Einbindung eigener Protokolle und Dienste ist MPEG zu starr und detailliert festgelegt. Gerade bei multimedialen Daten ist bei Speicherung und Transport durch die Vielzahl der konkurrierenden Ziele (Kosten, Qualität, Sicherheitsaspekte [10], [33]) die leichte, flexible und dynamische Austausch- und Konfigurierbarkeit von Mechanismen wichtig.

Die obigen Ansätze addressieren alle, auch die Themen der Zugriffsrechte auf und der Versionierung von multimedialen Inhalten. Die entsprechenden Lösungen greifen aber zu kurz. Um mit entsprechenden Diensten bei der Erstellung einer Applikation die Abstraktion von Problemen hinsichtlich Zugriffsrechten und Versionierung zu ermöglichen, muss eine Architektur durch einen einheitlichen Datenpfad durch entsprechende logische Schichten Rechte- und Versionssemantik garantieren können. Zum anderen müssen die entsprechenden Rechte- und Versions-Daten in expliziter Form vorliegen, um zuverlässig von spezifischer Applikationsinformation und damit -Semantik unterschieden werden zu können.

Zusammen mit der obigen Forderung nach der Bereitstellung dedizierter Speicher- und Transportdienste ergeben sich für eine Infrastruktur für verteilte multimediale Applikationen mit vielen Autoren zwei Forderungen, die über die vorhandenen Ansätze hinaus gehen:

1. eine Architektur, die nicht nur auf Retrieval ausgelegt ist, und die allgemeine Applikationsaspekte wie Rechte- und Versionsmanagement separieren kann, und
2. eine deklarative, explizite Datenstruktur, deren Anpassungen an die jeweilige Struktur flexibel, nachvollziehbar und möglichst modular sind.

Außerdem haben Szenarien, in denen Autoren kooperativ multimediale Inhalte erstellen und benutzen, weitere gemeinsame Aspekte, die auf einer höheren Abstraktionsschicht, als dies die obigen Ansätze bieten, kontrolliert oder zur Verfügung gestellt werden sollten. Beispiele hierfür ist vor allem die immer auftretende Spezialisierung der Autoren mit den daraus resultierenden verschiedenen Autorenrollen und die Organisation in Gruppen.

Kapitel 4 - Architektur und logische Datenorganisation

Nachdem Kapitel 2 das addressierte Anwendungsfeld beschrieb und in Kapitel 3 dazu existierende Lösungen betrachtet wurden, sollen notwendige Anpassungen dieser Lösungen und notwendige ergänzende Konzepte erarbeitet werden.

Die Anforderungen wurden dazu aus Anwendungssituationen abgeleitet, wie sie beim Erstellen gemeinsam genutzter multimedialer Lehrinhalte durch verschiedene Teams auftreten. Hierzu wurden Erfahrungen mit der Arbeit an Multimedia-Archiven aus mehreren Projekten verwendet. Die meisten davon wurden am oder unter Beteiligung vom Lehrstuhl KOM der TU Darmstadt durchgeführt (HyNode [9], Medianode [34], Multibook [46], Medibook [47] und kMed). Diese Projekte befassen bzw. befassten sich unter anderem mit der Struktur und Semantik wiederbenutzbarer multimedialer Lehrinhalte und erlebten dabei auch die Praxis der Erstellung solcher Inhalte. Als weitere wichtige Grundlage dienten Kenntnisse des Erstellens von Vorlesungsmaterial aus Erfahrungen des Teams der Vorlesungsersteller am Lehrstuhl Steinmetz. Tabelle 4 strukturiert dazu das Themenumfeld aus Kapitel 3. Die vertikale Schich-

Tabelle 4: Die im weiteren addressierten Probleme aus dem Themenumfeld

Organisation							Koordination, Notifikation	Administration unterstützende Rollen	
Benutzer	Hypertext	Transport, Metadaten, Ertweiterbarkeit	Manipulation von Dokumenten	Architektur Reflektion		Metadaten, Reflektion	Metadaten über Semantik, alternative Medien	Zugangsrechte, Metadaten, Versionierung	App.-spezifische Metadaten
Applikationsunterstützung	Erweiterbarkeit, Plugins	protocol plugins, application integration	Repräsentation, Speicherung	Ereignisse & Zustand Interoperabilität	Dienste (Namen, Persistenz,, Sicherheit)	QoS Synch.			z.B. Copyrights
Sitzung,, Transport			Streaming		Sitzung, Setup	QoS	Sperren von Zugriffem, Koordination	Autorisierung, Sitzungens	
Netzwerk					Adaption	Reservierung			
	Formatintegration	Protokollintegration	Medien	Verteilung	Setup, Anpassung	QoS	Konsistenz	Editieren	Applikationsspezifika

tung entspricht grob dem Abstraktionsgrad: jede höhere Schicht baut auf Diensten der darunterliegenden Schichten auf. Horizontal wurden verschiedene Aufgabenbereiche getrennt. Ein Ergebnis des letzten Kapitels ist, dass zusammenhängende Bereiche der Tabelle durchaus von vorhandenen Systemen abgedeckt werden (etwa die linke Seite von den Techniken des WWW). Besonders aber im rechten, oberen Bereich des Rasters, also bei abstrakten und inhaltsbezogenen Aufgaben, gibt es eine Vielzahl von separaten Teil-Lösungen. Die weitere Arbeit will in diesem Bereich ansetzen und als Beitrag Lösungen zu diesen Feldern (Koordination, applikationsspezifische Metadaten etc.) in einen zusammenhängenden Gesamtentwurf integrieren.

Die Bearbeitung in den weiteren Kapiteln besteht dann zum großen Teil aus der Anpassung und Integration bestehender Lösungen, die Zielapplikation beschreibt das Szenario eines verteilten Medienservers in Abschnitt 2.1.3.

Neben den funktionalen Zielen, die sich direkt aus der Zielapplikation ergeben, sind Flexibilität und einfache, planbare Erweiterbarkeit wichtige Nebenziele der vorgestellten Lösungen.

Dies soll zum einen laufende Anpassung an die schnell mutierenden Informationstechniken, vor allem im Bereich Multimedia und Kommunikation, ermöglichen. Zum anderen gewährleistet dies Anwendbarkeit über die direkte Zielapplikation hinaus.

Um die Nebenziele zu erfüllen, wird zunächst im folgenden Abschnitt eine Architektur erarbeitet, die die notwendigen Schnittstellen und Abstraktionen für die Erweiterungsmöglichkeiten bezüglich Multimediaformaten, Kommunikationsprotokollen und Plattformen bietet.

Die wichtigste Schnittstelle dieser Architektur, die logische Organisation des Anwendungsinhalts, wird dannach im Rest des Kapitels entwickelt.

4.1 Architektur

Im folgenden werden die notwendigen Abstraktionen für ein verteiltes multimediales System mit vielen Autoren in eine Architektur eingearbeitet. In einem ersten Schritt werden die Anforderungen und Ziele eingegrenzt, im zweiten Schritt eine entsprechende Architektur vorgestellt.

4.1.1 Anforderungen

Die wichtigsten Verallgemeinerungen eines solchen Programmes sind:

- Plattformunabhängigkeit, da viele Autoren verschiedene Plattformen für ihre verschiedenen Ziele benutzen werden,
- Unabhängigkeit von den sich ständig entwickelnden Medienformaten,
- Möglichkeit zur Integration verschiedener Zugangsprotokolle und -Programme (dedizierte Klienten, WWW-Klienten),
- Integration verschiedener Speichersysteme, da die sich entwickelnden Anforderungen von Multimediadaten von immer neuen, spezifischen Archivsystemen bedient werden,
- Integration von Semantik.

4.1.1.1 Plattform

Plattformunabhängigkeit wird gemeinhin erreicht, indem plattformspezifische Funktionalitäten identifiziert und in eine Einheit gekapselt werden.

Dies erlaubt es, plattformabhängigen Code gezielt durch Reimplementierung der Plattformabstraktion zu portieren.

Im Idealfall greifen alle anderen, plattformunabhängigen Komponenten über eine entsprechende Schnittstelle auf die Dienste dieser Einheit zu und sollten einfach durch Rekompilieren auf einer neuen Plattform zur Verfügung stehen. Eine andere Möglichkeit ist eine in plattformunabhängige Repräsentation und direkte Interpretation solcher Komponenten. Diese zweite Möglichkeit hat neben weniger Performanz den Nachteil, dass die Plattformabstraktion um mindestens einen kompletten, optimierten Interpreter erweitert wird.

Für diese Arbeit wurde der erste Ansatz gewählt: die Plattformabstraktion besteht aus einem kleinen Kern, der grundlegende Plattformfähigkeiten kapselt. Dazu gehört vor allem das Laden der Systemkomponenten, aber auch etwa der Zugriff auf lokale Dateien, auf lokale Einrichtingen wie etwa Netzwerkkarten oder das Verarbeiten von Betriebssystemsignalen.

4.1.1.2 Medienformate

Um nicht von Medienformaten abhängig zu sein, müssen alle möglichen Medienformate vom System behandelt werden können. Dies kann erreicht werden, indem jedes Medium vom System grundsätzlich als atomares Objekt, als *BLOB* (binary large object), behandelt wird, und die jeweiligen Erzeuger- und Verbraucherapplikationen (etwa Digitalkamera mit Encoder als Erzeuger, MPEG-Player als Verbraucher) als systemexterne Komponenten nur in Metadaten zu diesen Medien identifiziert werden. Um alternative Medienformate für denselben Inhalt verarbeiten und anbieten zu können, müssen die Metadaten der Medien insbesondere auch die semantische Äquivalenz von Medien angeben, etwa, wenn ein Video und ein Text beide dieselbe Erklärung desselben Sachverhaltes darstellen.

Um spezielle technische Anforderungen der Medien bezüglich Speicherung und Transport erfüllen zu können, müssen neben den Metadaten auch entsprechende Implementierungen von Speicherung und Transport ins System aufgenommen werden können.

Um neue Medienformate verarbeiten zu können, werden in die Architektur eingefügt:

- eine formale Beschreibung der Datenstruktur, um systemweit an einer Stelle das Format der Metadaten festlegen und erweitern zu können,
- eine Schnittstelle, um Datenimport- und Datenexportmodule zu integrieren,
- eine Schnittstelle, um Medien formatspezifisch speichern zu können (wie etwa auf einem Videoserver oder in einer SQL-Datenbank).

Nur die in die interne Datenstruktur zu integrierenden Metadaten sind eine spezifische Schnittstelle für die Abstraktion vom Medienformat. Die letzten beiden Punkte werden im folgenden für die Abstrahierung vom Zugang und von der konkreten Speicherung addressiert.

4.1.1.3 Zugang

Der Zugang zum System umfasst zum einen den Zugriff auf Inhalte und Metadaten, zum andern aber auch die Autentifikation von Autoren, die dem System bekannt sind, und der Integration von deren Arbeit.

Durch die Anzahl verschiedener Nutzer und Autoren und durch die Entwicklung muss Flexibilität gegeben sein, um hier gewünschte Zugangssysteme integrieren zu können.

Dazu wird eine Systemschnittstelle in die Architektur eingefügt, die die Integration Zugangskomponenten erlaubt, die Sitzungen (hier vor allem Autentifizierung) und den Import und Export von Daten implementiert.

Insbesondere muss systemextern jedes interne Datum eindeutig durch eine ID benannt werden können. Um die Kooperation von Autoren zu ermöglichen, muss den Nutzern eine möglichst einfache, gemeinsame logische Sicht auf die Systeminhalte präsentiert werden.

4.1.1.4 Speicherung

Die Vielfalt an Optimierungen, die für die Speicherung und den Zugriff auf Inhalte einer multimedialen Applikation möglich und notwendig sind, müssen ins System integriert werden können.

Um Optimierungen (Replikation, Auslagerung etc.) und eine möglichst einfache externe Nutzersicht zu ermöglichen, werden über die physikalische Speicherung nur die notwendigen Informationen (wie etwa Datenrate und Verzögerung beim Ausspielen) an den Benutzer weitergegeben. Dies erlaubt dem System eine möglichst weitgehende Autonomie, Entscheidungen über die physikalische Speicherung von Inhalten intern zu treffen.

Dazu wird eine Schnittstelle in die Architektur für Komponenten eingefügt, die die Implementierung oder Integration von Speicherdiensten zur Verfügung stellen.

4.1.1.5 Erweiterungen der Zugriffssemantik

Sind Module, die die Schnittstelle zur Speicherung von Daten implementieren, dazu da, das Universum der erreichbaren Speichertechniken und damit Inhalte zu vergrößern, ist es auch eine wichtige Funktionalität, diese Zugriffe mit Semantik anzureichern. Ein bekanntes Beispiel etwa aus Datenbanken und Dateisystemen sind Nutzerrechte, die Zugriffe einschränken oder verbieten können. Ein Beispiel, das spezifisch für multimediale Anwendungen ist, ist die die zeitliche Einschränkung oder Bestimmung von Zugriffen, um die Einzelmedien einer Präsentation sinnvoll synchronisiert zu exportieren. Ein weiteres Beispiel ist die Einschränkung von Zugriffen auf von der aktuellen, lokalen Plattform unterstützte Medientypen, welche auch alternative Medien nutzt, um Inhalte darstellen zu können.

Als Schnittstelle für Komponenten zur Erweiterung der Zugriffssemantik wird ein Filter definiert, der von allen Zugriffen der Zungangskomponenten benachrichtigt wird und diese blokkieren, verändern oder erlauben kann.

4.1.2 Realisierung

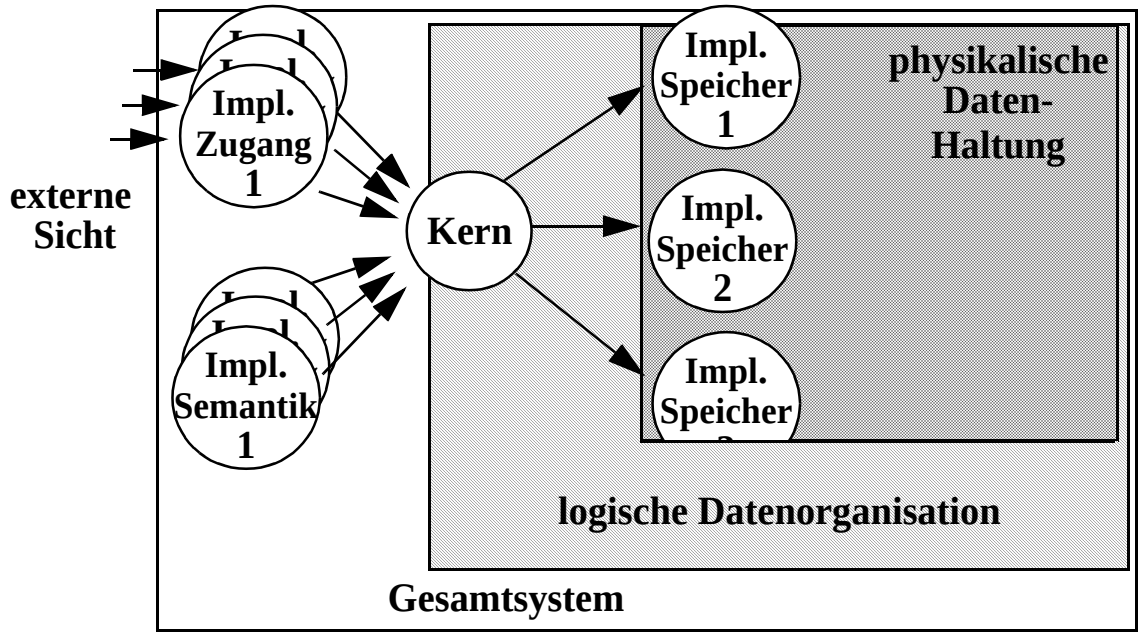

Abbildung 9: Kapselungen der Sicht auf die Datenhaltung

Abbildung 9 bietet eine Übersicht über die erforderlichen Schnittstellen, wobei die Beschreibung der Medien-Metadaten in der logischen Datenorganisation zu finden ist. Für die vorliegende Arbeit wurde eine direkte Umsetzung dieser Schnittstellen in eine Architektur gewählt (Abbildung 10). Die einzelnen Instanzen dieser Architektur sind dabei, abhängig von den integrierten Speicherkomponenten, sowohl als Peer-to-Peer-Instanzen (etwa für Replikationsdienste [38]), als Server (wenn etwa eine Videoserver-Komponente geladen ist) wie auch als Klienten (etwa, wenn sie SQL-Zugriffe nutzen) zu sehen.

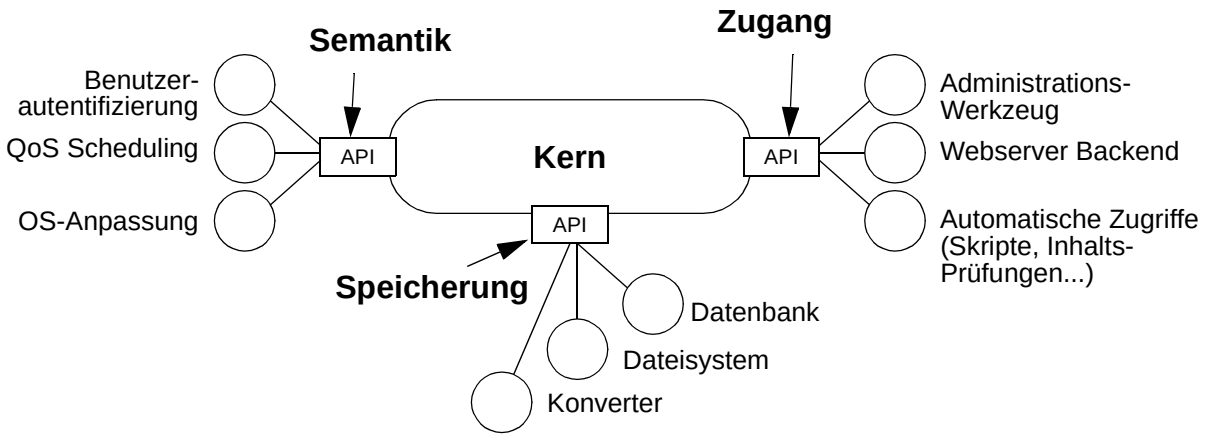

Abbildung 10: Architektur und Beispielmodule

Diese einfache Architektur erzeugt sehr mächtige, unspezifische Schnittstellen für Zugang, Zugangssemantik und Speicherung. Um dennoch betriebssicher und effizient operieren zu können, wird die Information über die Spezifika von Modulen und Inhalten in die Datenstruk-

tur aufgenommen, und gleichzeitig dem System die Möglichkeit zur Integration von Optimierungen auf tieferer Ebene (etwa Multicast beim Transport von Multimediadaten) zu erhalten.

Das bedeutet, dass sich die Komplexität der Applikation und damit des Zielsystems vor allem im Dateninhalt und seiner Struktur abbildet. Dies wiederum gibt dem System zur Laufzeit, dynamisch, Zugriff auf den Systemzustand. Desweiteren lassen sich damit durch weitgehend explizit deklarative, beschreibende Arbeit die Lösungen des resultierenden Systems auf andere Probleme anpassen.

Die restlichen Abschnitte dieses Kapitels erarbeiten eine Datenstruktur, die die Anforderungen des Vorlesungsarchivs aus Abschnitt 2.1.3 erfüllt. Im wesentlichen wird auch der restliche Teil dieser Arbeit, der die speziellen Ziele Zugriffsrechte, Zugriffsrollen und Versionierung addressiert, hauptsächlich diese Datenstruktur erweitern und entsprechende Semantik definieren bzw. implementieren.

4.2 Anforderungen an die Datenstruktur

Die interne Datenstruktur soll die logische Struktur aller Systeminhalte abbilden können, Medien und Organisationinformation genauso wie Information über Zustände.

Aus den notwendigen Schritten bei der Erstellung und Wiederverwendung von multimedialen Präsentationen ergeben sich Anforderungen an das System, welche nun in den Abschnitten 4.2.1 bis 4.2.7 kurz gesammelt werden. Neben diesen allgemeinen Anforderungen werden in den Abschnitten 4.2.8 bis 4.2.11 zusätzliche Anforderungen dargestellt, die sich aus technischen und organisatorischen Randbedingungen ableiten. Als letzter Unterabschnitt folgt eine Zusammenfassung der erarbeiteten Andorderungen an die logische Struktur.

4.2.1 Suchen und Sammeln von Inhalten

Ein wichtiger Teil der Arbeit bei der Erstellung und Wiederverwendung von Präsentationen besteht aus dem Suchen von Inhalten. Deswegen ist eine Annotation der Inhalte mit identifizierenden Metadaten bezüglich Inhalt und Thema, Format (Kodierung, Sprache), Erstellungsdatum und Autoren notwendig.

4.2.2 Import und Export von Inhalten

Es ist eine grundlegende Funktion, Inhalte ins System einfügen zu können und diese auch wieder außerhalb des Systems zugänglich machen zu können.

Hier ist zum einen der physikalische Transport wichtig: das System muss Protokolle unterstützen können, die Massendaten transportieren. Unmittelbar wichtig ist hier der Zugriff auf das Dateisystem auf dem lokalen Plattenplatz und etwa das File-Transfer-Protokoll FTP [132] für den Datenimport, das HTTP (Abschnitt 3.1.3).

Da der Bestand der Datenformate als erweiterbar und damit diese als grundsätzlich dem System unbekannt angenommen werden, muss auch die Möglichkeit zur Integration weiterer Transportmethoden vorgesehen werden. Ein Beispiel hierfür ist das Aufnehmen von Multimedia-Strömen via LC-RTP ([133], [62]), das zuverlässig Real-Time-Ströme transportiert. Hier-

aus folgt neben der Notwendigkeit einer Abstraktion des Zugangs in der Architektur (Abschnitt 4.1.1.3) die feine Einordnung der Medientypen, um die passenden Protokolle bestimmen zu können.

Neben dem physikalischen Transport existiert die Notwendigkeit eines logischen Zugangs: der Nutzer muss dem System die Inhalte nennen können, auf die er zugreifen will. Zum einen gibt es hier den direkten Zugriff und es gibt den Zugriff auf Daten, die von anderen Systeminhalten referenziert werden.

Für den ersten Fall sind Suchfunktionen, aber auch extern dauerhafte Addressen zum Zugriff auf gefundene Inhalte notwendig. Inhalte müssen innerhalb des Systems eindeutig benannt werden, also einen eindeutigen Identifaktor besitzen, mit obiger Forderung muss das System auch eine eindeutige und dauerhafte Abbildung dieser Identifikatoren auf externe Addressen besitzen.

Wenn Resourcen sich aufeinander beziehen (etwa durch Verweis, Enthaltung etc.), so muss beim Import bzw. Export jeder Verweis von einer externen Form auf den jeweilgen internen Identifikator abgebildet werden (bzw. vice versa). Aufgrund der dem System nicht transparenten Datenformate ist ein Mechanismus erforderlich, der diese Übersetzung ermöglicht.

4.2.3 Bearbeitung von Inhalten

Die Bearbeitung von Inhalten, die dem System nicht transparent sind (etwa Video-Formate) können nur durch die Arbeitsschritte Export, externe Bearbeitung, Import geschehen. Zu beachten ist hier, dass zum einen diese Folge von Arbeitsschritten durch die Realisierung von Zugangsmodulen vor dem Nutzer verborgen werden kann, die Editoren des jeweiligen Formats integriert präsentieren. Zum anderen ist zu beachten, dass beim Import die systeminterne Identität des Datums erhalten bleibt.

Der Import von Änderungen an Inhalten erfordert als weitere Unterstützung die Wahrung von Zugriffsrechten und die Verfolgung der Versionsgeschichte der Inhalte; diese werden im Abschnitt 4.2.6 und Abschnitt 4.2.7 weiter besprochen.

Nicht nur zur Ersparnis von Systemressourcen, vor allem zur Erhaltung der Konsistenz der Inhalte ist ein Kopieren von Inhalten nicht sinnvoll. Die Identität von verwendeten Ressourcen in verschiedenen Präsentationen ist eine wertvolle Information, die dem System erhalten bleiben soll. Mehrfachbenutzung von Ressourcen sollte also über mehrfache Referenzierung realisiertw werden.

4.2.4 Strukturierung von Inhalten

Inhalte müssen strukturiert werden können. Die generell übliche und bekannte Struktur ist die Hierarchie: die Zusammenstellung zu größeren Einheiten. Computer-Benutzer verwenden solche Strukturen überall dort, wo nicht formal hart strukturierte Inhalte gordnet werden sollen, die gewohnte Sicht darauf ist eine Baumdarstellung.

Im Umfeld multimedialer Inhalte ist noch interessant, dass optionale bzw. alternative Inhalte gegenüber notwendigen Inhalten gekennzeichnet werden können müssen, dass die Reihenfolge von Ressourcen, etwa Medien in einer Präsentation, von Bedeutung ist und dass es

neben der reinen Ordnungsstruktur auch ein Netz von Referenzen neben dieser Ordnung geben muss.

4.2.5 Trennung von Inhalten und Layout

Die Wiederverwendung von Inhalten in verschiedenen Organisationseinheiten macht es unabdingbar, Inhalte mit verschiedenen Layouts verbinden zu können. Dazu müssen Ressourcen, die Layouts spezifizieren, als solche gekennzeichnet werden. Auch müssen die Zusammenstellungen von Inhalten und Layouts als jeweils neue Einheiten, etwa Präsentationen, vom System behandelt werden.

Bei der Realisierung der Trennung von Inhalt und Layout ist zu beachten, dass Layouts durchaus auch Inhalte darstellen können, offensichtlich etwa in einer Vorlesung über Layout.

4.2.6 Rechtemanagement

Da von unterschiedlichen Autoren geistige Arbeiten vom System zur Verfügung gestellt werden können sollen, ist eine Kontrolle des Zugriffs auf diese Inhalte notwendig. Das Ändern und das Lesen von Inhalten darf nur unter Kontrolle des Systems geschehen, und die Besitzer bzw. Autoren von Inhalten müssen klare Vorgaben geben können, wer was mit ihren Werken machen darf. Daneben ist ein Rechtemanagement auch für die Betriebssicherheit eines solchen Systems notwendig: je komplexer und größer die Menge der Autoren ist, umso notwendiger ist eine Unterstützung der Organisation dieser Menge. Da das Rechtemanagement ein zentrales und großes Problem darstellt, wird es gesondert in Kapitel 5 behandelt.

4.2.7 Unterstützende Funktionalitäten

Zur Unterstützung der Arbeit vieler Autoren an multimedialen Inhalten sind einige Verinfachungen möglich.

Im Themenfeld der Datenbanken gibt es das Konzept der *Sicht* [24], die bei Zugriff auf einen Datenbestand neu organisiert und meistens auf Notwendiges verringert darbietet. Allgemeiner ist eine adäquate Präsentation mit der Einschränkung und Spezialisierung der Möglichkeiten zum Lesen und Ändern von Inhalten ein mächtiges Mittel zur Effizienzsteigerung und zur Vermeidung von Fehlern in der Arbeit von Menschen [19]. Eine mögliche Realisierung solcher Sichten in form von Rollen wird in Kapitel 6 präsentiert.

Notwendig ist eine Verfolgung der Historie der Inhalte durch das System: in der täglichen Arbeit von Autorenteams ist das Wissen über die Entstehung der Inhalte notwendig. Die Möglichkeit zum Zurücksetzen auf alte Versionen und zum Erstellen von Varianten, ohne diese Beziehung zwischen den entstandenen Ressourcen zu verlieren ist hilfreich. Versionierung wird als ein weiterer Schwerpunkt in Kapitel 7 addressiert.

Desweiteren muss das System offen sein, um automatisches Prüfen, Berichten oder Korrigieren von inhaltlichen und formalen Eigenschaften des Inhalts integrieren zu können. Wichtig sind hier Konsistenz- und Redundanz-Überprüfungen.

4.2.8 Quality of Service

Da Inhalte multimedial vorliegen können, sind die multimedialen Eigenschaften dieser Inhalte mit zu speichern und zu bearbeiten. Dies ist allerdings eine Teilfunktion der geforderten Mechanismen zum Annotieren von Inhalten (Abschnitt 4.2.1) und zum Im- und Export von Inhalten (Abschnitt 4.2.2), sofern diese nur die Repräsentation dieser QoS-Charkteristiken und deren Semantik realisieren.

4.2.9 Alternative Inhalte

Die semantische Äquivalenz von Inhalten ist eine Relation, die dem System mitgeteilt werden können muss und die es benutzen muss.

Semantische Äquivalenz von zwei Inhalten bedeutet, dass diese Inhalte dasselbe bedeuten und sich gegenseitig ersetzen können. Wichtig ist ihre Beachtung deswegen, weil bei multimedialen Inhalten die Repräsentation eines Sachverhaltes in einer sehr ressourcenverbrauchenden Form wie etwa als Video oft gewünscht ist, aber nur nach Abgleich mit der Menge der vorhandenen Systemressourcen einer schmaleren Repräsentation, etwa als Bild oder gar Text, vorgezogen werden kann.

4.2.10 Signalisierungsweg für Systemdienste

Die systeminterne Datenstruktur muss Inhalte mit Information über Nutzer (etwa beim Rechtemanagement) und mit Information über das System selbst in Beziehung setzen können. Dazu sollte mindestens ein passender Auschnitt aus diesen organisatorischen Informationen in der Datenstruktur vorkommen. Wenn die Repräsentation dieser Daten uniform mit der Repräsentation des sonstigen Inhalts ist, und zum Beispiel die Addressierung denselben Formalismen folgt, kann auch ihre Verarbeitung dieselben Mechanismen benutzen, die zum Verteilen und Speichern der Inhalte benutzt werden. Dies erlaubt es, Diensten wie zum Beispiel der physikalischen Replikation oder Verlagerung von Inhalten, zur Signalisierung den (zumindest logisch zentralen) Datenbestand des Systems zu benutzen.

Es ist wichtig, dass verschiedene Typen von Daten verschieden physikalisch gespeichert und transportiert werden können. Offensichtlich ist dies für solche verschiedenen Datentypen wie Texte und Videos. Im Kontext von Signalisierung ist aber hervorzuheben, dass auch Metadaten, also Attribute und Strukturen der Systeminhalte, eine eigene Kategorie von Daten darstellen mit zwar kleinem Umfang, welche aber am schnellsten von allen Inhalten systemweit zur Verfügung stehen müssen.

4.2.11 Integration mit physikalischen Gegebenheiten

Die Datenstruktur muss das System dort unterstützen, wo keine ideale systemweite Verteilung von Inhalten und Zuständen möglich ist, Maßnahmen zur vermeidung und Korrektur von Konflikten und Inkonsistenzen vorzunehmen. Dies betrifft etwa Attribute, die bearbeitete Inhalte vor einer parallelen Bearbeitung schützen, oder andere Hilfsinformationen speichern.

4.2.12 Zusammenfassung

Im Einzelnen sind diese Anforderungen und Randbedingungen:

- Sammel- und Suchfunktionen nach Inhalten
- Zugriff auf und Verfügbarkeit aller Inhalte
- Import und Export von komplexen bzw. einfachen, ungeordneten bzw. geordneten Inhalten in unterschiedlichen Formaten
- Cut/Delete, Copy, Paste von Inhalten, Layouts, Präsentationen und Informationen
- Rechtsschutz / Zahlungsmöglichkeit von zur Verfügung gestellten Inhalten
- Erstellung von Präsentation und Strukturierung von Präsentationselementen
- beliebige, einfache Kombinationen von Inhalten und Layout
- Aktualisierung, Updates und unterschiedliche Versionen
- Verwertung von Layoutvorlagen und ausführbaren Programmen auch als Inhalt
- Hilfsmittel / Vereinfachungen
- Berücksichtigung von Quality of Service
- Ausgabe einer fertigen Präsentation
- betriebssystemunabhängige Implementierung
- offene Standards zur flexiblen Schnittstellenverwaltung
- Präsentationsarchiv
- Anwendung in der Praxis muss auf Professoren abgestimmt sein

Im folgenden Kapitel 4.3 wird eine Datenstruktur vorgestellt, die diese Anforderungen erfüllt.

4.3 Konzeption der Datenstruktur

Auf Grundlage der Anforderungen aus dem vorigen Abschnitt werden in diesem Abschnitt Entscheidungen über die interne Datenstruktur getroffen. Einige dieser Entscheidungen ergeben sich zwingend aus den Anforderungen, andere wurden aus den im folgenden dargestellten Gründen getroffen.

4.3.1 Zentrale Archivierung

Um eine konsistente und gemeinsame Sicht aller Autoren auf die Inhalte zu realisieren, wird ein einziger, globaler Systeminhalt implementiert. Das bedeutet, dass die physikalische Verteilung der Inhalte vor dem Benutzer verborgen wird. Interessant ist für ihn die Verfügbarkeit (bzw. die QoS-Parameter der Verfügbarkeit) von Daten. Diese Eigenschaften sind der einzige Bezug zur physikalischen Repräsentation der Inhalte, den das System dem Benutzer erlaubt.

4.3.2 Trennung von Inhalt, Layout und Struktur

Die Trennung von Inhalt, Layout und Struktur ist die zentrale Eigenschaft der Datenorganisation. Inhalt sind die tatsächlichen Daten der benutzten Texte, Bilder bzw. Medien. Das Layout beschreibt die Darstellung der Inhalte, und in der Struktur wird die Ordnung der Inhalte hergestellt. Die Kombination von strukturiertem Inhalt mit verschiedenen Darstellungsformen wird damit sehr erleichtert. Die Vor- und Nachteile der Trennung werden im folgenden aufgeführt.

Nachteile:

- Vorarbeit der Autoren beim Einfügen des Inhalts ist notwendig,
- viele Konvertierungsprobleme nicht kombinierbarer Formate.

Vorteile:

- flexible Kombinierung von Inhalten und Layout,
- Redundanzfreiheit und Konsistenz durch getrennte Strukturspeicherung,
- wird von verschiedenen Standards unterstützt.

Eine andere Möglichkeit ist ein fester Regelkatalog für die Benutzung von Werkzeugen zu Präsentationserstellung oder eine zentrale Sammlung von Konvertierungsprogrammen.

Die Nachteile der Trennung sind nicht zu unterschätzen. Durch die vorgesehene Benutzung bereits existierender Präsentationen wird eine nicht unerhebliche Arbeit an Konvertierungen notwendig. Nur bei neuem Inhaltsmaterial kann das Einfügen der Inhalte mit relativ wenig Arbeitsaufwand erfolgen.

Die Trennung von Inhalt, Layout und Struktur wird in der Organisation der Daten durch die Elemente Ressourcenbaum, Layoutliste und Präsentationsliste erzielt, die weiter unten beschreiben werden.

4.3.3 Der Ressourcenbaum

Die Trennung von Inhalt, Layout und Struktur soll in der Datenorganisation sowohl graphisch, als auch in der Speicherung selbst erfolgen. In dem Bereich für Inhalte (dem Ressourcenbaum) kann jede mögliche Form des Inhalts un- und vorstrukturiert gespeichert werden. Diese Inhalte werden als Ressourcen bezeichnet. Eine Ressource kann ein Monomedium oder ein Multimedium sein. Eine Ressource kann demnach sowohl ein atomarer Inhalt, als auch ein sehr komplexer und weiter unterteilter Inhalt sein.

Die grundlegende Ressource ist ein blob (*binary large object*). Diese Ressourcen sind dem System nicht transparent und erhalten ihre Semantik erst durch die Verwendbarkeit in entsprechenden externen Applikationen. Blobs können sein: Textblöcke, Graphiken, Audio- und Videodaten, Java Applets und Applikationen. Auch sogenannte Stylesheets, das sind Layoutvorlagen, sind Blob-Ressourcen. Vorstellbar sind z.B. Powerpoint Formatvorlagen, aber auch Cascading Style Sheets (CSS) und XML Stylesheets (XSL). Um eine als Ressource abgespeicherte Layoutvorlage als Layout nutzen zu können, muss sie in der Layoutliste referenziert werden (siehe Kapitel 4.3.4).

Der Ressourcenbaum ist eine strenge Hierarchie. Damit wird eine Baumstrukturierung der Ressourcen erreicht. Am Beispiel von Dateisystemen und deren graphischer Darstellung ist diese Baumstrukturierung zu begründen. Aber nicht nur die bekannten Darstellungsmöglichkeiten eines Baumes führten zur Entscheidung. Auch kann durch die damit verbundene Vorstrukturierung der Ressourcen eine Automatisierung ermöglicht werden (siehe Kapitel 4.3.7).

4.3.4 Die Layoutliste

Die Layoutliste ist eine Zusammenstellung aller in dem Ressourcenbaum verfügbaren Ressourcen, die als Layoutvorlagen dienen können.

Eine Layoutvorlage ist die Beschreibung eines Layouts in einem bestimmten Layoutformat (z.B. Power Point Vorlage oder ein CSS Stylesheet).

Die Zusammenstellung erfolgt durch Referenzen auf die Layoutvorlagen. Diese Referenzen werden in der Layoutliste gespeichert und angezeigt. Zusätzlich werden einer Layoutreferenz weitere Informationen als Metadaten (siehe Kapitel 4.3.6) zugeordnet. Diese Informationen werden z.B für die automatische Präsentationserzeugung benötigt (siehe Kapitel 4.3.7).

Da die Layoutvorlagen auch als Ressourcen im Ressourcenbaum abgelegt werden, bestehen für sie die gleichen Möglichkeiten zur Vorstrukturierung wie für andere Ressourcen.

Layoutvorlagen können wieder aus verschachtelten Layoutvorlagen für Untereinheiten einer Präsentation (etwa Doppelstunden als Teile einer Vorlesungsreihe) bestehen. In der Layoutliste können dann unabhängig voneinander die verschiedenen Ebenen der Verschachtelung referenziert werden. In einer Präsentation, die diese Layoutvorlage benutzt, können zusätzliche Layoutreferenzen die vorgegebenen Layoutvorlagen überschreiben. Einer Folie kann zusätzlich eine ganz bestimmte Folienlayoutvorlage zugewiesen werden.

Andere Möglichkeiten, die die Strukturierung der Layoutvorlagen innerhalb des Ressourcenbaums betreffen, sind alternative Medien (siehe Kapitel 4.3.8) oder multimediale Referenzen (siehe Kapitel 4.3.9).

4.3.5 Die Präsentationsliste

In der Präsentationsliste werden alle Präsentation gespeichert und bearbeitet. Die endgültige Struktur der Inhalte wird in der Präsentationsliste hergestellt.

Um eine Präsentation zu erstellen, muss eine Layoutreferenz aus der Layoutliste mit Ressourcen aus dem Ressourcenbaum verknüpft werden. Die Struktur der Präsentation wird dann manuell oder automatisch in der Präsentationsliste erstellt. Über Strukturierungsfunktionen können die Ressourcen in der Präsentation bearbeitet werden. Die verschiedenen Automatisierungsmöglichkeiten werden in Kapitel 4.3.7 besprochen.

Innerhalb der Präsentationsliste ist es möglich, eine Präsentation mit unterschiedlichen Layoutreferenzen mehrfach zu verwenden. Denkbar sind z.B. unterschiedliche Layouts für Handout und Internet Varianten einer Präsentation.

4.3.6 Metadaten zur Dokumentation

Zu Teilen der Anforderungen, die innerhalb dieser Arbeit diskutiert werden, existieren Lösungen aus anerkannten Standards zum Annotieren von Multimedia-Daten (etwa MPEG-7 [23]) und zur Annotation von Lehr- und Lernmitteln (etwa LOM [130] [50]). Hier wird ein Ausschnitt aus LOM verwendet, der für die vorliegenden Ziele ausreicht. Dieser umfasst Autor, Entstehungsdatum, Kommentar, Titel und Sprache einer Ressource.

4.3.7 Dynamische Erzeugung von Präsentationen

Ein ehrgeiziges Ziel der Datenorganisation von Medianode ist die dynamische Erzeugung von Präsentationen. Diese Automatisierung umfaßt die automatische Auswahl und Aufteilung von Ressourcen auf ein Layout anhand verschiedener Metadaten. Dabei ist hier nicht die automatische Platzierung, wie etwa in Textsatzsystemen, im Fokus.

Schwerpunkt ist hier, die grundsätzliche Eignung des resultierenden Systems zur dynamischen Erzeugung von Präsentationen. Genutzt werden soll diese Fähigkeit, um Monomedien nach technischen Gesichtspunkten zu kombinieren, etwa nach benötigter Bandbreite oder zu unterstützendem Medientyp.

Bei einer Automatisierung erfolgen iterativ die wiederholte Auswahl und Aufteilung von Ressourcen auf das entsprechende Layout. Die automatische Auswahl von Ressourcen kann nach verschiedenen Kriterien erfolgen. Abgesehen von weniger komplexen Kriterien, z.B. Auswahl nach Autor oder Datum, ist die Automatisierung auch für die komplexe Auswahl von Ressourcen nach vorhandener Ausgabemöglichkeit vorgesehen. Da die Zuordnung eines Layouts und die Kontrolle des Resultats einen signifikanten Anteil der Arbeit an einer Präsentation einnehmen kann, ist eine Unterstützung dieser Arbeitsschritte ebenso wichtig wie die Unterstützung der Pflege der Ressourcen. Die Pflege der Zuordnungen von Ressourcen zu Layouts erfolgt über die Liste der Präsentationen, die genau diese Zuordnungen abbilden sollen.

In der Organisation der Daten im Medianode System werden Informationen über Ausgabequalitäten und -quantitäten in den Metadaten ’device’ und ’dimension’ gespeichert. Eine device ist eine Ausgabeform, z.B. HTML oder Text. Eine dimension ist z.B. Dauer eines Videos oder eines Tondokumentes.

Jeder Ressource und jeder Layoutreferenz sind Devices und Dimensions als Metadaten zugeordnet. Im Ressourcenbaum ist aus den Werten dieser Devices und Dimensions zu erkennen, was von einer Ressource benötigt wird, um diese Ressource in einer Präsentation verwenden zu können.

In der Layoutliste ist aus den Werten der Devices und Dimensions zu erkennen, was ein Layout anbietet, um bestimmte Ressourcen darstellen zu können.

Der Mechanismus ordnet die Ressourcen in der vorstrukturierten Reihenfolge auf die Layoutvorlage. Es wird dabei überprüft, ob die ausgewählten Formate der Ressourcen mit der Layoutvorlage zusammenpassen oder ob es geeignete Konvertierungsprogramme oder -module gibt. Im Zweifel werden die Alternativen ausgewählt, die am besten mit der Layoutvorlage zusammenpassen. Die einzelnen Ressourcen werden auf die verschiedenen Präsentationsfolien verteilt und der gesamte Verbrauch der ausgewählten Ressourcen wird mit dem gesamten

Angebot der Layoutvorlage verglichen. Beim Überschreiten des Angebots wird die Präsentation im vollständig automatisierten Verfahren um die unwichtigen Ressourcen gekürzt.

Wichtig für die Güte einer Präsentation ist die sorgfältige Pflege und detaillierte Benutzung der Metadaten. Je besser und sinnvoller die Metadaten sind, desto weniger Nachbearbeitung wird notwendig sein. Die Nachbearbeitung erfolgt auf manuellem Wege. Wie bei der reinen manuellen Präsentationserzeugung sollte es möglich sein, bestimmte Automatisierungsmechanismen abschalten zu können oder zumindest Dialoge oder Meldungen vorzusehen. Sollen Präsentationen nach nicht-technischen Vorgaben zusammengestellt werden, etwa nach Schwierigkeitsgrad der einzelnen Inhalte [48], muss ein Datenschema benutzt werden, das die entsprechenden Metadaten beinhaltet.

Da Autoren manuell Einfluss auf das Resultat ihrer Arbeiten nehmen wollen, ist eine Mischung aus manuellen und automatischen Verfahren notwendig. Dies wird so angeboten, dass ein Mechanismus ausgelöst werden kann, der rekursiv eine Zuordnung von Ressourcen zu Teil-Layouts erstellt, manuelle Zuordnungen aber mit höherer Priorität versieht. Im Wechsel von manuellen und automatischen Zuordnungen für eine (Teil-) Präsentation ergibt sich so eine Struktur, die soweit wie möglich flexibel ist, also vom System an manuelle Änderungen durch automatische Zuordnung der nicht manuell zugeordneten Ressourcen angepasst, gleichzeitig aber beliebig genau durch manuelle Zuordnungen festgelegt werden kann.

4.3.8 Alternative Medien

Die Verwendung und Kennzeichnung alternativer Medien ist ein weiteres wichtiges Ziel der Datenorganisation. Die Auswahl alternativer Medien erfolgt etwa dann, wenn der ursprünglich ausgewählte Inhalt nicht dargestellt werden kann. Gründe können im Fehlen eines geeigneten Darstellungsprogramms, aber auch in einer zu schwachen Netzwerkleistung liegen.

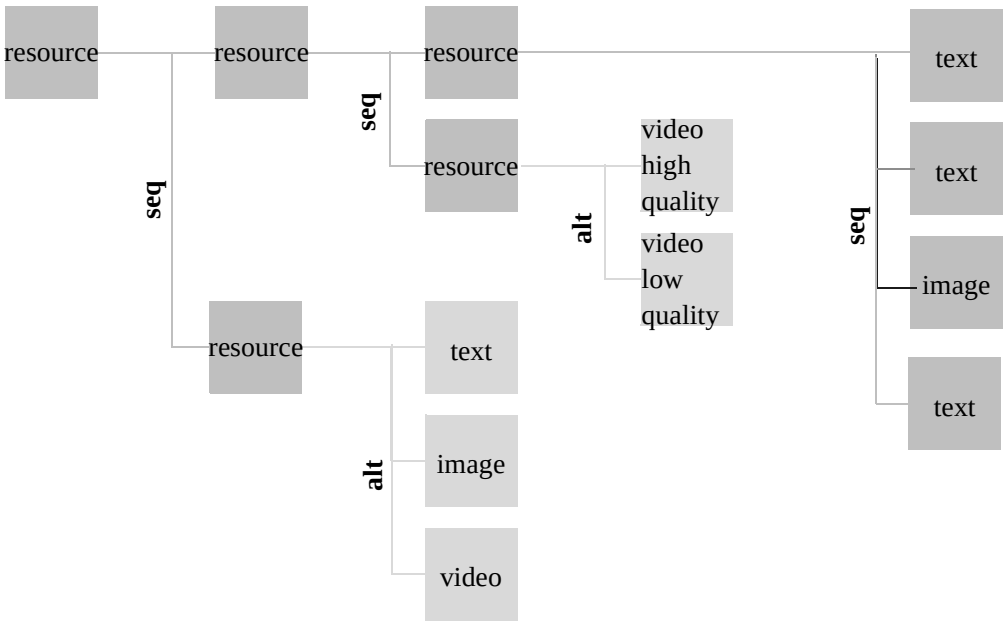

Abbildung 11: Alternative Medien und Sequenzen im Ressourcenbaum

Neben der mehrstufigen Vorstrukturierung durch die Baumstruktur, können die Knoten einer Ebene alternativ und sequentiell im Verhältnis zueinander stehen. Alternativ stehen zwei oder mehrere Ressourcen zueinander, wenn sie den gleichen Inhalt haben, aber sich z.B. in Darstellungseigenschaften voneinander unterscheiden. Bei einer automatischen Erzeugung einer Präsentation (siehe Kapitel 4.3.7) wird anhand von Darstellungseigenschaften des Ausgabemediums auf alternative Ressourcen zurückgegriffen.

Zum Beispiel benutzt jeder Web-Server das Konzept alternativer Medien, wenn die aufgerufene Seite nicht vorhanden ist oder der Web Server überlastet ist. In beiden Fällen wird anstatt der gewünschten Seite ein Dokument mit einer Fehlermeldung angezeigt.

Die Eigenschaft, dass Ressourcen alternativ, also verschieden, aber inhaltlich äquivaltent sind, ist verschieden von der Eigenschaft, notwendige Untermedien innerhalb eines Multimediums zu sein. Dennoch lassen sich beide durch hierarchische Zusammenfassung wie in Abbildung 11 darstellen. Im ersten Fall, in dem eine Alternative (im Gegensatz zu einer Sequenz) von Untermedien repräsentiert werden soll, ist die Reihenfolge nicht relevant, da andere (Meta-) Daten über die Auswahl entscheiden.

Die Möglichkeit Ressourcen ungeordnet abzulegen ist trotz der beiden Möglichkeiten der alternativen und der sequentiellen Speicherung immer noch gegeben. Da die eigentliche Strukturierung erst in der Präsentationsliste erfolgt, sind die besprochenen Strukturierungsmöglichkeiten der Baumstruktur und der alternativen und sequentiellen Speicherung nicht zwingend. Nur bei der automatischen Erzeugung wird auf diese Vorstrukturierung zurückgegriffen.

4.3.9 Multimediale Referenzen

Ein weiteres Mittel zur Vorstrukturierung verschiedener Inhalte stellen multimediale Referenzen dar. Eine Art der Referenz, die des Enthalten-Seins, wird durch den hierarchischen Aufbau des Inhalts schon explizit dargestellt.

Weitere Arten von Referenz sind notwendig, um eine echte Wiederbenutzung zu ermöglichen - im Gegensatz zur Kopie und der danach getrennten Bearbeitung von Inhalten.

Da sich die Arten von Referenzen aus den verwendeten systemexternen Erstellungs- und Präsentationsformaten und -Anwendungen der Inhalte ergeben, ist hier nur ein flexibler, erweiterbarer und Mechanismus sinnvoll. Entsprechende Beobachtungen und Feststellungen enthält etwa die Spezifikation von XLink [134]. Die im Weiteren gewählte Darstellung von Referenzen lehnt sich vereinfachend an XLink an.

4.3.10 Konvertierungsmodule

Eine Konvertierung von Medien tritt beim bei Import und Export von Präsentationen und Ressourcen auf. Monomediale Ressourcen können als dem System intransparente, atomare Einheiten behandelt werden, deren interne Struktur nur von entsprechend einzubindenden Modulen bei Import und Export behandelt wird.

In multimedialen Ressourcen müssen bei Import und Export die Referenzen auf die importierte bzw. exportierte Instanz der Zielmedien umgestellt werden. Die dazu notwendige Infor-

mation wird im System abgelegt (s. auch Abschnitt 4.1.1.1). Entsprechendes gilt für die Struktur von Präsentationen bei der Ausgabe der erstellten Layouts.

Konvertierungsmodule werden jeweils für neue Medienformate und -applikationen erstellt und integriert.

4.3.11 Plattform

Die Organisation der Daten wird möglichst betriebssystemunabhängig implementiert. JAVA eignet sich durch seine Platformunabhängigkeit dafür. Außerdem soll bei der Realisierung möglichst auf bereits existierende Produkte zurückgegriffen werden. Als Format der Datenorganisation wird deshalb XML verwendet.

Zu beachten ist, dass betriebssystemspezifische Funktionen in den Programmiersprachen C/C++ implemtentiert werden. Durch die Benutzung offener Standards werden die Schnittstellen zwischen den Teilen sehr flexibel gehalten und die Erweiterbarkeit gegeben.

Nachteile von JAVA:
- langsamer Parser
- der gewählte System-Kern ist in C/C++ (Abschnitt 4.1.1.1) realisiert, der Übergang vom Kern zum Parser ist damit komplex

Vorteile von JAVA:
- plattformunabhängig
- Integration Java mit XML / XSL ist weit fortgeschritten
- Klassen für alle Schnittstellen vorhanden
- wichtige Klassen sind open Source

Nachteile von XML:
- alle freien Tools sind z.Zt. noch um Betastadium
- DOM Generierung sehr langsam

Vorteile von XML:
- lesbar
- offener Standard
- basiert auf Trennung von Struktur, Inhalt und Layout
- Export in andere Formate über XSL sehr leicht
- Verschachtelung innerhalb des Dokumentes möglich
- viele Open Source Tools

4.3.12 Liste der zentralen Entscheidungen

Die folgende Liste fasst noch einmal die zentralen Entscheidungen für die Organisation der Daten zusammen, die aus den dargelegten Gründen getroffen wurden.
- zentrale Archivierung in der Datenbank
- klare Trennung von Inhalt, Layout und Struktur
- Baumstruktur zur Speicherung und Darstellung von Inhalten
- Liste zur Zusammenstellung und Darstellung der Layouts

- Speicherung der Layouts als Ressourcen im Ressourcenbaum
- endgültige Strukturierung der Inhalte und Übersicht über Präsentationen in der Präsentationsliste
- Metadaten für Informationen über jedes Element
- Suchfunktionen über Metadaten
- Automatisierung der Präsentationserstellung über Metadaten
- Rechte- und Versionsmanagement
- Alternative Medien zum Ausgleich fehlender Darstellungs- und Zugriffsmöglichkeiten
- multimediale Referenzen
- Konvertierungsmodule für Import und Export
- graphische Benutzeroberflächen für Autoren, Designer und Professoren
- Platformunabhängigkeit durch JAVA, XML
- Entschärfung der Schnittstellenproblematik durch die Wahl offener Standards

Im folgenden Kapitel wird der technische Rahmen der Implementierung dargestellt. Die Elemente des Konzepts der Organisation der Daten werden mit den entsprechenden technischen Komponenten verbunden.

4.4 Realisierung der Datenstruktur

Aus den vorgesehenen Eigenschaften der Datenorganisation wird im folgenden Kapitel der technische Rahmen für eine Implementierung dargestellt. Dabei wird ein Überblick über die wichtigen Implementierungsbereiche und die verwendbare Software gegeben.

4.4.1 Technisches Konzept

Das technische Konzept ist aus folgenden Elementen aufgebaut:

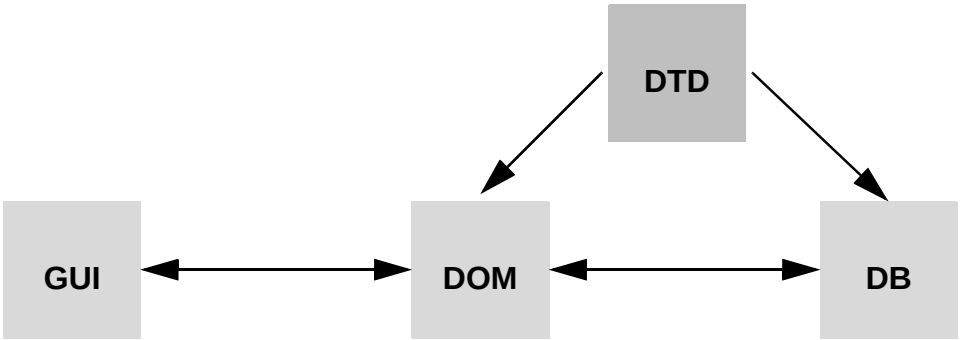

Abbildung 12: Technisches Konzept der Datenorganisation im Überblick

- DTD für Struktur von Datenspeicherung, -verwaltung und -transport
- DOM zur Verwaltung der Daten
- DB für zentrale Speicherung der Daten
- GUI für Anzeige und Bearbeitung der Daten

Eine Implementierung der Datenorganisation kann in drei Bereiche unterteilt werden:

- Schnittstellen des Datenorganisation
- Struktur der Datenspeicherung
- Funktionalität

Diese Bereiche sind eng miteinander verknüpft. Es würde zu erheblichen Implementierungsproblemen führen, diese Bereiche völlig getrennt voneinander zu betrachten.

Als Programmiersprache für die Schnittstellen und die Funktionalität des Systems ist JAVA vorgesehen. Durch die Wahl von JAVA als Programmiersprache und XML als Austauschformat ist es möglich, die Organisation der Daten in eine einheitliche Softwareumgebung mit offenen Schnittstellenstandards einzubetten. Dadurch ist es möglich, JAVA im nachhinein durch C bzw. C++ zu ersetzen (siehe Kapitel 4.4.2 und Kapitel 4.5).

Im folgenden werden Übersichten über die drei Bereiche Schnittstellen, Struktur und Funktionalität gegeben. Die einzelnen Teilbereiche werden identifiziert und Implementierungslösungen vorgestellt.

4.4.2 Schnittstellen des Datenorganisation

Für die Datenorganisation müssen verschiedene Präsentationen und die entsprechenden Transformation zwischen diesen Präentationen erstellt werden. Beispiele sind etwa eine graphische Präsentation zur Manipulation, eine textuelle Darstellung zur Übermittlung und eine tabellarische zur Speicherung. Weil für eine komplexe hierarchische Datenorganisation wie die hier entstehende viel Unterstützung in form von Implementierungen, Tools, Parsern und Übersetzern besteht, wurde eine Darstellung der Daten (-organisation) als XML-Dokument (DTD) als Ansatz gewählt.

Abbildung 13 identifiziert die einzelnen Schnittstellen der Datenorganisation, abgebildet auf entsprechende XML-Techniken. Das document object model - DOM - bildete dabei das zentrale Element der Datenorganisation.

Die Schnittstellen zur umrandeten XML Komponente sind für die Beispielimplementation wichtig. Das Auslagern der Daten in einer XML Datei ist nur interessant, solange die DOM/DB und die GUI/DOM Schnittstellen nicht funktionsbereit sind.

4.4.2.1 DB <-> DOM

Diese Schnittstelle umfasst folgende Funktionalitäten:

- Auslesen der gesamten Datenbank ins DOM
- Speichern eines kompletten DOMs in der Datenbank
- Auslesen von einzelnen Daten aus der Datenbank
- Speichern einzelner Daten in der Datenbank

JDBC und ein XML Parser in Java sind die Schnittstellen vom DOM zur Datenbank. Diese Schnittstellen sind lokal auf Funktionalität getestet und implementiert worden.

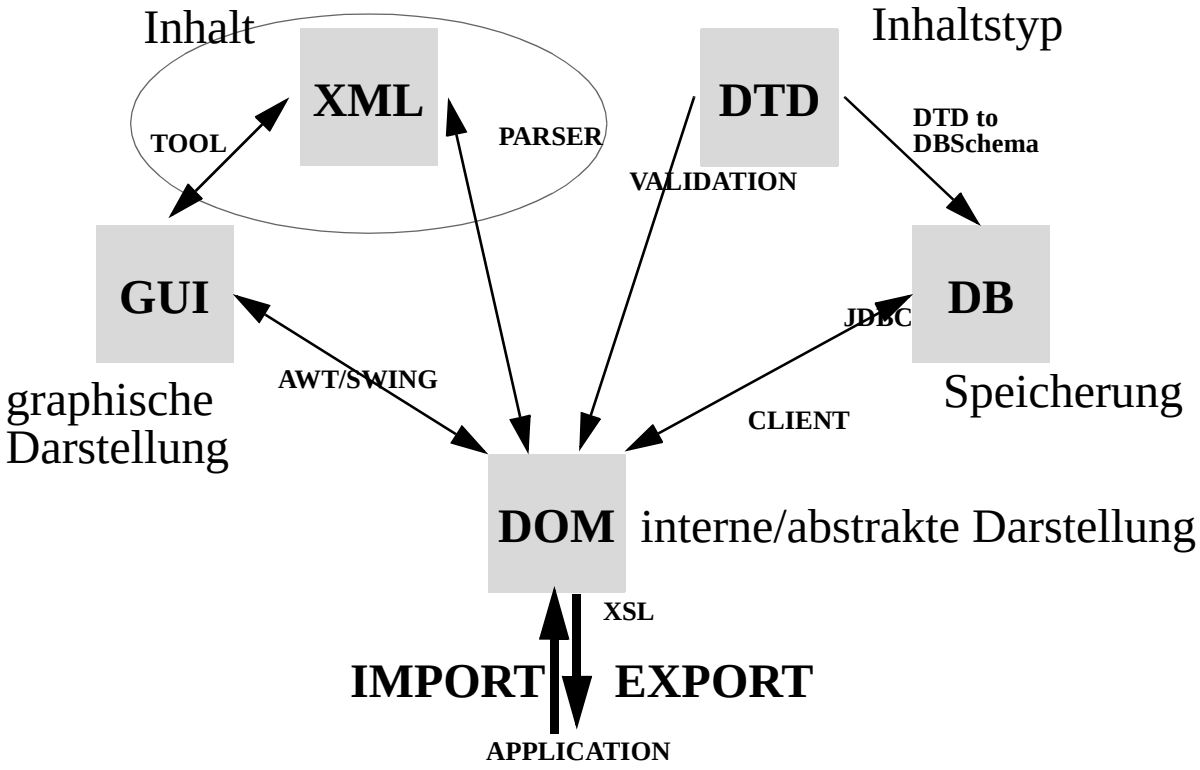

Abbildung 13: Schnittstellen des Datenformats

4.4.2.2 GUI <-> DOM

Die Schnittstelle zwischen dem DOM und einer graphischen Benutzeroberfläche soll folgendes ermöglichen:

- Darstellung des DOMs
- Darstellung einzelner DOM Unterbäume
- Speicherung von Veränderungen im DOM

Diese Schnittstelle wurde noch nicht implementiert. Allerdings ist es möglich, den Umweg über eine XML Datei zu gehen. Dabei wird das DOM als XML Datei ausgeschrieben und vom einem XML Editor geladen (siehe Kapitel 4.4.2.3 und Kapitel 4.4.2.4).

4.4.2.3 GUI <-> XML

Diese Schnittstelle wurde bereits auf Funktionalität getestet. Als Test - GUIs kommen viele XML Editoren in Frage. Die gewünschte XML Datei wird meistens als Baum aufgebaut. Selbst die Bearbeitung und Speicherung der XML Datei über eine GUI ist möglich.

Als GUI wurde der XML Editor Xeena von IBM verwendet [128].

Der verwendete XML Editor Xeena hat Schwächen, die eine zukünftige Erstellung einer eigenen GUI nützlich erscheinen lassen:

- kein Open Source, d.h. kein direkter Zugriff auf die Schnittstelle GUI <-> DOM
- Erweiterung der Funktionalität nur schwer möglich

4.4.2.4 XML <-> DOM

Fast alle Java-XML-Parser bauen aus einer XML Datei ein DOM auf. Das Einlesen und die Ausgabe zwischen XML Datei und DOM erfolgt dabei über einfache Befehle. Als Java-XML-Parser kann z.B. Xerces von Apache [129] oder xml4j von IBM Alphaworks verwendet werden.

4.4.2.5 Import -> DOM

Der Import von Präsentationen und Dateien ist ein Hauptbestandteil der Datenorganisation. Folgende Importtypen können dabei unterschieden werden:

- Text, Metadaten
- alle BLOBs (Bilder, Videos etc.)
- bestehende Präsentationen

Die Eingabe von Text oder Metadaten in eine XML Datei kann über einen normalen Texteditor oder einen XML Editor erfolgen. Über einen XML Editor wird das DOM allerdings direkt verändert. Da die Schnittstelle zur Datenbank über das DOM implementiert wird, ist der Import in die Datenbank hier ohne weiteren Zwischenschritt möglich. Allerdings ist eine flüssige Arbeit mit XML Editoren zur Zeit noch nicht gegeben. Die Benutzung eines Texteditors ist aus Geschwindigkeitsgründen vorzuziehen.

Der Import größerer Texte, BLOBs oder bestehender Präsentationen muss über zusätzliche Funktionalität erfolgen. Die gewünschte Datei wird dabei über eine Pfadangabe mit der XML / DOM verbunden. Eine Möglichkeit besteht darin, die Datei von dem Programm direkt in die Datenbank schreiben und die Pfadangabe durch eine Referenz in die Datenbank ersetzen zu lassen. Die andere Möglichkeit ist eine Textkodierung der Datei in XML / DOM, was allerdings zu einer nicht geringen Aufblähung von XML / DOM führt.

Eine bestehende Präsentation kann über eine manuelle Aufteilung der einzelnen Elemente in die XML Datei eingefügt werden. Eine automatische Aufteilung ist noch zu implementieren.

4.4.2.6 DOM -> Export

Fertige Präsentationen können aus dem DOM zur Ansicht exportiert werden. Dabei sind folgende Anforderungen zu beachten:

- Export übers Netz oder auf Festplatte
- dynamische Erzeugung der einzelnen Folien
- einmalige Erzeugung aller Folien
- dynamischer Export der BLOBs aus der Datenbank

Die Umwandlung der Präsentationen aus der DOM in ein entsprechendes Format wird über Layout und Konvertierungsmodule gelöst.

Auch diese Schnittstelle wurde in der Beispielimplementierung gelöst. Allerdings mit einigen Einschränkungen, so wurden die Bilder in einem Dateisystem und nicht in der Datenbank referenziert.

4.4.2.7 DTD -> XML

Die DTD wird benötigt, um Regeln bezüglich der Struktur des DOMs aufzustellen. Sowohl die verwendeten XML Editoren, als auch die verfügbaren Java Klassen für XML sehen eine automatische Überprüfung der Struktur der XML Datei anhand der DTD vor.

Diese Schnittstelle ist in dieser Form einsetzbar.

4.4.2.8 DTD -> DB Schema

Werkzeuge, um aus einer DTD automatisch ein DBSchema zu generieren, wurden nicht integriert. Diese würden dass die Erstellung der Tabellen der Datenbank automatisieren, zusätzlich würde diese Verbindung das Aus- und Einlesen von dem DOM in die Datenbank stark vereinfachen.

4.4.3 Datentyp

Der hier vorgestellte Datentyp wurde im System Medianode erfolgreich verwendet [34] und ist auch Grundlage für die Spezifikation oder Realisierung der Lösungen in den weiteren Kapiteln.

Im Anhang ist eine formale Datentypdefinition (DTD [135]) für einen Datentyp angegeben, der die oben geforderten Eigenschaften und Möglichkeiten besitzt; in den folgenden Unterabschnitten werden spezielle Aspekte des Datentyps aufgezeigt.

4.4.3.1 Globale Verwaltungsbereich

Abbildung 14 zeigt den globalen Verwaltungsbereich der Datenorganisation. Der ’mediarug’ ist der Wurzelknoten der Datenorganisation und beinhaltet damit das gesamte System inkl. aller Daten.

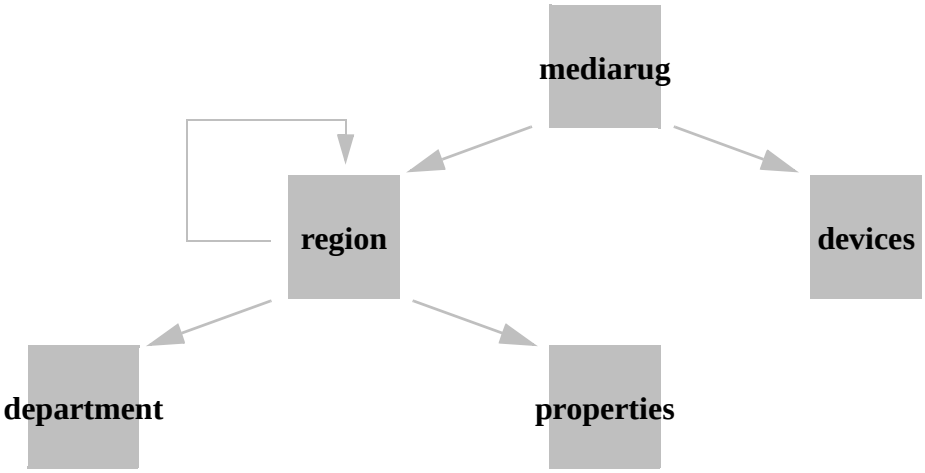

Abbildung 14: Oberste Ebene der Datenorganisation

An dem Wurzelknoten ’mediarug’ hängen verschiedene Regionen mit Eigenschaften. Diese Eigenschaften beinhalten z.B. den Namen des Systemverwalters für diese Region und die Liste aller in dieser Region für Aktionen berechtigten Benutzer und Benutzergruppen. An jeder Region können wiederum andere Regionen (Unterregionen) mit Eigenschaften hängen. Wenn eine Region keine untergeordneten Regionen besitzt, wird ihr ein ’department’ zugeordnet, in dem Inhalte und Metadaten gespeichert werden.

Im globalen Verwaltungsbereich werden auch die ’devices’ und ’dimensions’ abgelegt.

4.4.3.2 Inhaltsbereich

Abbildung 15 gibt eine Übersicht über ein ’department’ - d.h. den lokalen Inhaltsbereich - einer Region.

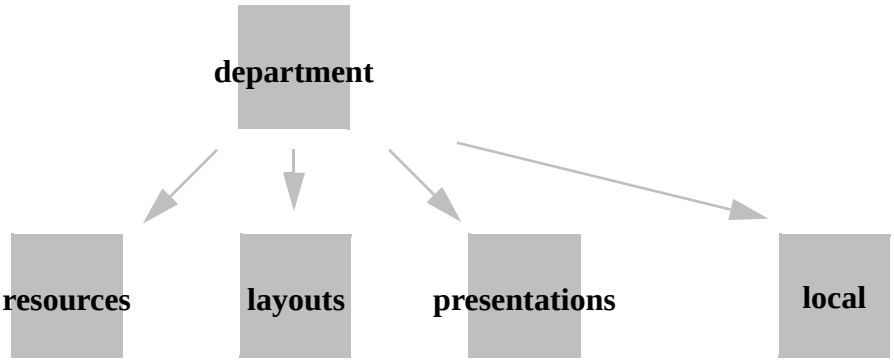

Abbildung 15: Inhaltsbereich des Datenorganisation

Klar zu erkennen ist die Dreiteilung in Inhalte (’resources’), Layouts und Präsentationen (siehe Kapitel 4.3.2 bis Kapitel 4.3.5). In dem local Bereich werden zusätzliche Information - z.B. die lokal zur Verfügung stehenden Rechner - gespeichert.

Abbildung 16 gibt eine Übersicht über die Struktur des Ressourcenbaums.

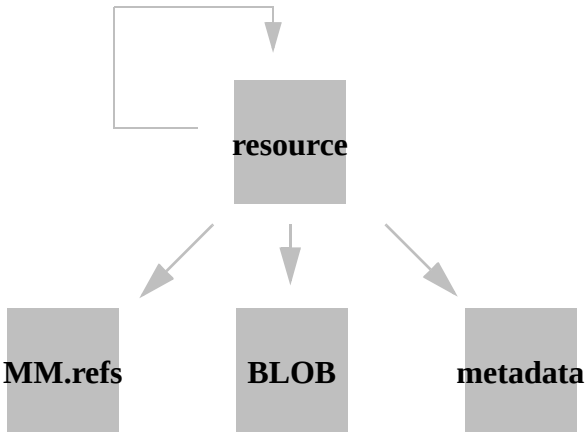

Abbildung 16: Die Struktur des Ressourcenbaums

Ressourcen können beliebig viele Ressourcen enthalten. Dadurch wird die in Kapitel 4.3.3 beschriebene Baumstrukturierung der Inhalte möglich. Die multimedialen Referenzen (siehe Kapitel 4.3.9) sind optional. Das BLOB-Element ist der eigentliche Datenspeicher einer Ressource. Hier können Text und MM-Inhalte als Text encodiert gespeichert werden. Die Metadaten sind jeder Ressource zwingend zugeordnet.

Während die multimedialen Referenzen in der Beispielimplementierung nur angedacht wurden, ist eine Speicherung von Inhalten in den BLOBs und von Metadaten bereits möglich.

Abbildung 17 gibt eine Übersicht über die Struktur der Layoutliste.

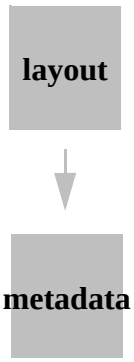

Abbildung 17: Die Struktur der Layoutliste

Jedes Layout referenziert eine Layoutvorlage aus dem Ressourcenbaum (siehe Kapitel 4.3.4). Die Metadaten sind jedem Layout zwingend zugeordnet.

Der Mechanismus wurd im System *medianode* auf der CeBIT 2000 mit einfachen Layouts demonstriert, später wurde die Benutzung von Layouts durch eine erweiterbare Beschreibung von Ausgabemedien verbessert, vornehmlich durch die Datenstrukturen ’devices’ und ’dimensions’ (vrgl. Anhang C).

Abbildung 18 gibt eine Übersicht über die Struktur der Präsentationsliste.

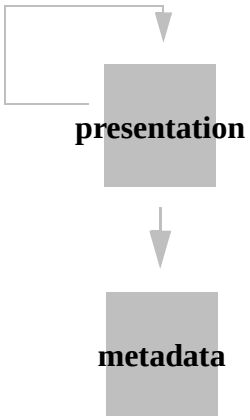

Abbildung 18: Die Struktur der Präsentationsliste

Jede Präsentation referenziert Inhalte aus dem Ressourcenbaum (siehe Kapitel 4.3.5). Durch die Möglichkeit der Verschachtelung der Präsentationen kann z.B. eine Doppelstunden / Folien Aufteilung erfolgen. Die Metadaten sind jeder Präsentation zwingend zugeordnet.

Die Präsentationsliste wird in dieser Form in der Beispielimplementation verwendet.

4.5 Zusammenfassung

In diesem Kapitel wurde eine einfache Architektur und ein detaillierter Datentyp vorgestellt, die zusammen eine mögliche Plattform zur Realisierung von verteilten multimedialen Systemen mit vielen Autoren darstellen.

Als wichtigste Grundlage bei der Realisierung dienten die Werkzeuge, die im Umfeld der XML-Standards zur Verfügung standen.

Neben seiner Funktion, die Arbeit der nächsten Kapitel an Lösungen zur Versionierung, zu Rechten und zu Rollen zu unterstützen, produzierte dieses Kapitel eine wiederverwendbare Grundlage auch für andere als die addressierte Applikation. Diese Wiederverwendbarkeit resultiert aus der Flexiblität, erweiterbar und anpassbar zu sein. Die Verwendung einer zentralen formalen Spezifikation (DTD) im Zentrum des Konzepts formalisiert Erweiterungen und Anpassungen des Datentyps, was solche Anpassungen explizit und damit sicherer macht.

Kapitel 5 - Rechtemanagement

Multimediale Inhalte stellen Werte dar. Die Zugriffsmöglichkeiten müssen in einem Mehrbenutzersystem eingeschränkt und gesteuert werden können, um diese Werte zu schützen. Schutz ist notwendig gegen böswillige, unerlaubte Nutzung wie auch gegen ungewollte, fehlerhafte Veränderungen beim Pflegen multimedialer Inhalte [27]. Gängiges Mittel, um solches zu erreichen, sind (Zugriffs-) Schutzmechanismen in Verbindung mit Benutzerauthentifizierung.

In diesem Abschnitt wird der Aspekt Sicherheit im Hinblick auf die Besonderheiten beim Zugriff auf und der Pflege von großen multimedialen Applikationen betrachtet.

Genauer: ein System wird vorrausgesetzt, das

1. Autoren sicher identifiziert,
2. geschlossene Sitzungen mit den Autoren realisiert, um Inhalte und Aktionen sicher zuordnen zu können,
3. die Integrität und Authentizität von Daten innerhalb des Systems gewährleistet,
4. systemweit konsistent auf denselben Daten arbeitet.

Mittel, um obige Forderungen zu erfüllen, sind Sicherheitsinfrastrukturen zum Authentifizieren (etwa eine Public key Infrastructure) und Protokolle für die gesicherte Kommunikation in Sitzungen (etwa SSL). Der dritte obige Punkt kann durch Implementierung eines eigenen, geschlossenen Systems realisiert werden, etwa [14], oder durch eine Schicht von Kontrollmechanismen, die über einer unsicheren Infrastruktur eingesetzt werden. Der vierte Punkt ist in einem wirklich verteilten und skalierbaren System nicht zu erreichen. Diese Idealisierung soll vorläufig dazu dienen, die Inferenzen zu ermitteln, die die speziellen Eigenschaften des Anwendungsfeldes *multimediale Anwendung mit vielen Autoren* mit Aspekten der Sicherheit hat.

Es soll hier ermittelt werden, welches spezielle Ziele und erwünschte Eigenschaften eines Rechtemanagements für ein großes, verteiltes multimediales System sind, und es werden detaillierte Vorgaben auf logischer Ebene (auf Applikationsebene) gegeben, um solche Ziele zu erfüllen.

Die Art der Strukturierung (Benutzungsbeziehungen, etwa Hyperlinks, zwischen Medien), die Art der Arbeitsvorgänge und die Art der zu ermöglichenden Nutzungen wird hier untersucht Es werden generische Festlegungen und Lösungen bezüglich Rechte-Vererbung, Struktur der Zugriffsdefinitionen (ACLs) erarbeitet. Diese werden für den Inhaltstyp “Lehrinhalte für Universitätsvorlesungen” konkretisiert.

Der Umgang mit fehlerhaften oder unvollständigen Daten und damit die Aufgabe der Idealisierung der perfekten Verteilung ist nicht Zielrichtung dieser Arbeit. Dies wird, wie auch die Versionierung, auch aufbauend auf den hier präsentierten Ergebnissen in anderen Arbeiten behandelt [38].

5.1 Anwendungsbeispiele

Durch folgende Anwendungsszenarien sollen typische, notwendige und charakterisierende Eigenheiten eines Rechtemanagements für die Erstellung multimedialer Anwendungen durch viele Autoren identifiziert werden.

Als Rahmenszenario dient das Vorlesungsarchiv aus Abschnitt 2.1.3: es ist das komplexeste der drei vorgestellten Anwendungen, vor allem wegen der dezentralen, aber zusammenhängenden Organisation seiner Nutzer in einzelnen Lehrstühlen.

Daneben aber bietet dieses konkrete Beispiel die Möglichkeit der Evaluierung von sonst abstrakt bleibenden Konzepten.

5.1.1 Urheberrechte

Eine Anforderung an das neue Rechtemanagementsystem ist die Unterstützung urheberrechtlicher Aspekte.

Die herrschende Rechtslage in Deutschland sieht vor, dass nicht schon eine Idee ein schutzfähiges Gut gemäß dem Urhebergesetz ist, sondern erst die Verwirklichung einer Idee, die zusätzlich bestimmte Voraussetzungen wie Individualität, eigenständige Struktur, etc. erfüllen muss. Durch eine Verwirklichung, die den eben genannten Anforderungen entspricht, entsteht ein “urheberschutzfähiges Werk”.

Der Ersteller eines urheberschutzfähigen Werkes, welches sich im vorgestellten System als Resource niederschlägt, erwirbt durch die Erstellung die Urheberrechte (eigentlich Urheberpersönlichkeitsrechte) an diesem Werk. Verbleiben diese Rechte beim Urheber, so spricht man von Verwertungsrechten, gehen sie auf eine andere Person über, so spricht man hingegen von Nutzungsrechten. Der Ersteller ist berechtigt, die Nutzung seines Werkes jemand anderem entgeltlich oder unentgeltlich zu gestatten, also diesem anderen Nutzungsrechte einzuräumen. Außerdem kann er alle seine Verwertungsrechte an eine andere Person veräußern, die dieser Person dann als Nutzungsrechte zur Verfügung stehen. Trotzdem verbleiben in besonderen Konstellationen einige unveräußerliche Rechte bei dem Ersteller.

Die Urheberrechte können auch bei einer Gruppe von Benutzern liegen, wobei prinzipiell die Gruppenmitglieder dem Umfang ihres Beitrages nach berechtigt sind.

Die Nutzungsrechte lassen sich nach verschiedenen Kriterien charakterisierern. Zum einen kann man zwischen unbeschränkten und beschränkten Nutzungsrechten unterscheiden. Unbeschränkt bedeutet hier, dass zwar eine Nutzungsart definiert wurde, aber nicht der Umfang dieser Nutzung (zeitlich, quantitativ, etc.). Beschränkt bedeutet folgerichtig, dass neben der Nutzungsart auch der Umfang der Nutzung definiert wurde. Die zweite Unterscheidung betrifft das ausschließliche bzw. das einfache Nutzungsrecht. Ausschließliches Nutzungsrecht bedeutet, dass der zur Nutzung Berechtigte, und nur der zur Nutzung Berechtigte, das Werk nutzen darf und kann. Einfaches Nutzungsrecht hingegen bedeutet für einen zur Nutzung Berechtigten, dass er das Werk zwar nutzen, aber nicht davon ausgehen kann, dass er der einzige ist, der dieses Werk nutzt.

5.2 Zentrale Entscheidungen

Das vorgeschlagene Rechtemanagementsystem soll speziell an die Anforderungen des Medianodeprojektes angepaßt werden. Dabei werden die am besten geeigneten Ansätze der betrachteten vorhandenen Rechtemanagementsysteme benutzt, aber auch neue Ideen verwirklicht. Dieses Kapitel soll die Entscheidungsfindung kurz darstellen und sie somit nachvollziehbar machen.

5.2.1 Ziele

Zugriffsschutz für Inhalte

Ein wesentlicher Aspekt im Umgang mit wertvollen Inhalten ist naturgemäß die Möglichkeit einer wirksamen Zugriffskontrolle, um wertvolles Wissen nur berechtigten Personen zugänglich zu machen. Das WWW und sein Protokoll HTTP allein stellen dies nicht nicht zur Verfügung. Der Zugriffsschutz muss der Anwendungsanforderung entsprechende Zugriffsarten unterscheiden, mindestens Lesen und Schreiben.

Weiterhin müssen, soweit nötig oder nützlich, die Zugriffsschutzmechanismen die strukturellen Besonderheiten von multimedialen Daten berücksichtigen. In einfachen hierarchischen Dateisystemen gibt es als Strukturelemente nur Verzeichnis, Datei und einfache Metadaten (Besitzer, Änderungsdatum, Größe) mit einer Aggregationsrelation. Inhalte einer multimedialen Applikation mit Medieninformation, Systeminformation, Nutzerinformation und einem erweiterten Metadatensatz und verschiedenen Relationen (inhaltliche Äquivalenz, Hypertext-Referenzen und Einbindung von Medien) ergeben feinere und andere Granularität als Dateisysteme (s. Abschnitt 4.3).

Applikationsspezifika

Desweiteren stellt der Fokus *multimediale Applikation* eine Spezialisierung gegenüber generellen Archiven und Speichern dar. Das bedeutet, dass mehr Merkmale der Nutzung bekannt sind und unterstützt werden sollten. Dazu gehören generell urheberrechtliche Aspekte. Bei der Nutzung und Archivierung von Medien sind immer Informationen und Mechanismen bezüglich Urheberschaft, Besitzer, Copyright und Nutzungsrechten bzw. deren zeitliche Grenzen und Preis relevant.

Die Bearbeitung von multimedialen Inhalten ist strukturiert: oft wird die Bearbeitung der Inhalte von der Bearbeitung der Bearbeitung der Präsentation dieser Inhalte getrennt, personell oder zeitlich (vrgl. Abschnitt 6.2). Auch eine (thematische) Aufteilung in Teams ist durch die aufwendige Pflege multimedialen Inhalts Standard für große Anwendungen. Die Zusammenarbeit hier ist aber weniger strikten Regeln unterworfen wie für streng formale Inhalte, etwa Programmtexte in CVM Werkzeugen oder Workflow in WFM-Systemen (Abschnitt 3.2).

Einfach, erweiterbar, klar

Der Grad der Nutzung eines Systems ist in einem hohen Maße davon abhängig, wie einfach es benutzt, ergänzt und/oder aktualisiert und verwaltet werden kann. Dies gilt auch für ein Rech-

temanagement, welches durch seine Konzeption auf den besonderen Anwendungsbereich ausgerichtet werden kann und soll. Durch ein sinnvolles Rechtemanagementsystem wird die Verwaltung eines Systems transparent und dadurch auch die Akzeptanz des Systems erhöht.

Da die Bearbeitung multimedialer Inhalte durch viele Autoren schon inhärent komplex ist, darf das Rechtemanagement, soweit es nicht vereinfacht ist, nur möglichst wenig erweitert werden. Das bedeutet, dass möglichst wenige Konzepte und Regeln auf möglichst anwendungsnahe Art defniniert werden. Anwendungsnah bedeutet hier, dass sie nach einer Analyse der Zielapplikation möglichst einfach im Problembereich selbst identifiziert werden können und damit den im Applikationsbereich arbeitenden Nutzern schon möglichst explizit bekannt sind.

Konzepte und Regeln sollten weiterhin klar unterscheidbar sein. Implizite Effekte sind sinnvoll für eine effizientere Formulierung von Regeln, aber nur so, wie es absolut üblich, wohldefiniert und auch wohlverstanden ist. Dies trifft auf die hierarchische Vererbung von Rechten zu.

Dezentrale Administrierbarkeit

Besonders die physikalische, aber auch die organisatorische Skalierbarkeit einer Anwendung mit vielen Nutzern in vielen Funktionen macht eine dezentrale Administrierbarkeit notwendig.

Neben einer generellen Struktur, oft eine Hierachie, die eine organisatorische Gliederung ermöglicht, gehören dazu auch spezifische Vorgaben, die die Eigenschaften der Anwendung spiegeln, etwa die Autonomie von Verwaltungseinheiten (s. Abschnitt 6.1.1 bis Abschnitt 6.1.3).

5.2.2 Konsequenzen für das zu wählende Rechtemanagement

Benutzer und Gruppen

Die Elemente eines Rechtemanagementsystems sind teilweise vorgegeben. Im vorgestellten System werden natürlich die Benutzer die herausragende Stellung einnehmen. Daneben existieren, ebenso wie in AFS und NT, Gruppen von Benutzern, die für die Rechtevergabe und somit zur Verwaltung im System benutzt werden können. Die Administration von Gruppen und Benutzern soll im neuen System der Einfachheit halber und aus Gründen der Verwaltbarkeit nur den Administratoren obliegen. Damit geht das vorgeschlagene System mit seiner strengen Arbeitsteilung einen anderen Weg als AFS, wo die Gruppenverwaltung den Benutzern obliegt. Ein Unterschied besteht auch zum Windows NT System, wo auch die Hauptbenutzer zur Benutzer- und Gruppenverwaltung berechtigt sind.

Ressourcen

Bei den bereits vorgestellten Systemen wird meistens zwischen Verzeichnissen (oder Collections) und Dateien unterschieden. Diese Unterscheidung wird im neuen System aufgrund der multimedialen Inhalte aufgehoben. Es wird nur noch von Ressourcen gesprochen. Da eine Ressource mehrere Kindressourcen haben kann, ist es möglich, dass sie, ähnlich wie im UNIX-System, die Funktion eines Verzeichnisses übernimmt.

Diese Regelung wurde getroffen, um eine optimale Anpassung des vorgeschlagenen Systems an die Anforderungen eines Rechtemanagementsystems für Multimediadokumente zu erreichen. Dabei wurde insbesondere auf Einfachheit geachtet. Ferner musste der gewählte Ansatz allen Fallbeispielen entsprechen und alle Lehrstuhldiskussionen überstehen.

Weitere Elemente des vorgeschlagenen Systems werden die Hierarchieebenen sein. Eine detaillierte Beschreibung findet sich in Kapitel 4.

Insgesamt ist noch anzumerken, dass alle Elemente des vorgeschlagenen Systems auf ein Grundschema zurückführbar sind, was die Implementierung in einer objektorientierten Programmiersprache erleichtern soll.

Eine weitere Entscheidung, die getroffen wurde, betrifft die Identifizierung der Elemente des Rechtemanagementsystems. Die Elemente sollten aus erkennbaren Gründen auch identifizierbar bleiben, wenn ihr Name oder ihre Eigenschaften geändert wurde. Es muss also eine Art “Primärschlüssel” geben, der nicht vom Namen des Elements abhängig ist. Hier bietet sich eine Regelung ähnlich der SID des NT-Systems an.

Hierarchische Struktur

Die Anwendung erweist sich schon in der Analyse als in vielem hierarchisch strukturiert. In Abschnitt 4.3 wurde unter anderem deswegen eine streng hierarchische Struktur präsentiert als adäquat für eine Multimedia-Anwendung mit vielen Autoren beschrieben.

Auch in Systemen, die ein Rechtemanagement erfordern, sind Hierarchien ein wichtiges Element, um die Rechteinformation übersichtlich zu strukturieren (s. Abschnitt 3.2). Insbesondere das zentrale Ordnungskonzept in allen diesen Systemen die Vererbung von Rechten ist, ohne die auch schon sehr einfach strukturierte Dinge wie Verzeichnisse mit Dateien nicht mehr mit dem Benutzer zumutbarem Aufwand verwaltbar wären.

Es wird im vorgeschlagenen System die Möglichkeit geben, Standorte und Abteilungen darzustellen und sich auch selbst verwalten zu lassen. Dabei wird auf eine strenge Arbeitsteilung geachtet. Das bedeutet, dass Administratoren von höheren Ebenen regelmäßig keine administrativen Aufgaben in untergeordneten Ebenen wahrnehmen können und sollen.

Access Control Lists

Eine weitere Entscheidung betraf die Art der Zugriffskontrolle, hier wurde den von den UNIX-Systemen, WebDAV und AFS, sowie NFS bekannten ACLs (Access Control Lists) der Vorzug gegeben. ACLs lassen sich leicht realisieren, verwalten und anwenden. Hinzu kommt die Erfahrung mit dem Umgang, den die Administratoren und Benutzer mit den ACLs aus den genannten Systemen bereits haben.

Die neu vorgeschlagenen Rechte, die sich in den ACLs anwenden lassen, orientieren sich stark an den bereits vorgestellten Systemen. Die Rechtedefinition wird jedoch (wie das gesamte System) offen gehalten, um eventuell später auftretenden Anforderungen gerecht werden zu können. Unterschiede wird es jedoch in der Vererbung und der jeweiligen Anwendung der definierten Rechte geben. Im neuen, streng hierarchischen System soll der Benutzer oder die Gruppe, welche eine Ressource verwalten darf, auch sämtliche Kindressourcen direkt verwalten dürfen. Dazu werden im vorgeschlagenen System die ACLs in eine GRANT- und

DENY-Sektion aufgeteilt, die jedoch unterschiedlich behandelt werden. Die Rechte, die in der DENY-Sektion einer ACL einer Ressource aufgeführt sind, werden für den entsprechenden Benutzer oder die entsprechende Gruppe verboten, und zwar den gesamten Hierarchiebaum abwärts. Versucht ein Benutzer nun auf eine Ressource zuzugreifen, so muss der Hierarchiebaum untersucht werden, ob dem Benutzer das entsprechende Zugriffsrecht entzogen wurde. Anders hingegen sieht es mit der GRANT-Sektion aus. Diese muss anders behandelt werden, da es später möglich sein soll, einem Benutzer oder einer Gruppe auf einer bestimmten Ressource ein Recht zu gewähren, auf einer anderen Kindressource das Recht zu entziehen um es auf einer Kindeskindressource wieder zu gewähren. Hier wird ein neuer Mechanismus benutzt, nämlich der, dass nach einem GRANT-Recht in der jeweils assoziierten ACL gesucht wird. Wird ein Eintrag gefunden, der auf den entsprechenden Benutzer zutrifft, so wird dieses Recht angewandt. Kann kein Eintrag gefunden werden, so wird in der ACL der Elternressource nach solch einem Eintrag gesucht. Bezüglich der GRANT-Sektionen gilt also das erste Auftreten eines Benutzers oder einer Gruppe, in der der Benutzer Mitglied ist. Eine formellere Beschreibung mit einem einfachen Beispiel findet sich in Kapitel 4.

Durch die beschriebenen Mechanismen bzgl. GRANT- und DENY-Sektion wird die Funktionalität zum Verwehren des Zugriffs wie sie im AFS-System besteht hier, explizit genutzt und mit der Flexibilität der Gewährung des Zugriffs aus dem NT-System kombiniert.

Besitzrechte

Ein Konzept, welches ebenfalls leicht abgewandelt von den existierenden Systemen AFS und NT übernommen wird, ist dass der Ownership. So hat der Besitzer einer Ressource grundsätzlich alle Rechte auf die Ressource und die ACL der Ressource. Ein großer Unterschied ist allerdings die Vererbung der Besitzeigenschaft. So hat der Besitzer der Elternressource im vorgeschlagenen System auch alle Möglichkeiten auf alle Kindressourcen (außer Änderung der Besitzeigenschaft). Damit wird auf den späteren Anwendungszweck eingegangen. Im Medianodesystem sollen stets dem “Chef” eines Ressourcenbaums alle Rechte eingeräumt werden, da er sich für das jeweilige Projekt verantwortlich zeichnet. Übernommen von Windows NT wurde prinzipiell die Veränderung der Besitzeigenschaft, die die Administratoren an andere Benutzer oder Gruppen übertragen können. Im Unterschied zu NT kann diese Eigenschaft an andere übertragen und nicht übernommen werden. Dies entspricht dem Anspruch nach Trennung zwischen Administration und Inhaltsbearbeitung.

Neu ist ebenfalls, dass die Änderung der Besitzeigenschaft nicht nur nachvollzogen werden kann, sondern stets mit einer aussagekräftigen Systemmeldung verbunden sein wird (s. Kapitel 4).

Charging und Copyright

Eine Anforderung, an das neue System bestand darin, dass der Zugriff auf Ressourcen auch kostenpflichtig gestaltet werden sollte. Um dies zu erreichen, konnte auf keines der bestehenden Systeme zurückgegriffen werden. Die Entscheidung fiel zugunsten einer Lösung mittels ACLs aus, da sich dadurch sehr einfach unterschiedliche Kosten für Benutzer und Gruppen erreichen lassen (s.auch Kapitel 4). Ebenfalls dort wird es ein neues Element geben, nämlich

das Ablaufdatum für Rechteeinträge. Dieses neue Element stellt eine einfache Realisierung eines “Copyright” dar.

5.3 Vorgeschlagenes Rechtemanagementsystem

5.3.1 Konzepte

Vorgeschlagen wird ein hierarchisches System mit Vererbung, in dem die eigentliche Rechtevergabe wie in bereits betrachteten Systemen über ACLs geregelt ist. Hierbei werden etablierte und bewährte Konzepte übernommen und, wo notwendig angepasst. In Kapitel 6 werden zusätzlich zu den ACL-Rechten werden noch Rollen definiert, welche die bereits vergebenen Rechte über einen anderen Ansatz einschränken, um Fehlbedienungen vermeiden zu helfen.

5.3.1.1 ACLs

Die Rechtevergabe im vorgeschlagenen System wird über ACLs erfolgen. Dabei ist es möglich Benutzer oder Gruppen in ACLs zu verwenden, die aus einer GRANT und einer DENY Sektion bestehen werden. Wie der Name bedeutet, werden Rechte, die in der GRANT-Sektion aufgeführt werden, dem Benutzer oder der Gruppe gewährt, Rechte, die in der DENY-Sektion aufgeführt sind, werden entzogen. Wird einem Benutzer oder einer Gruppe ein Recht auf eine Elternressource eingeräumt (GRANT), so vererbt sich dieses Recht auch auf die Kindressource, es sei denn, in der GRANT-Sektion der Kindressource ist etwas anderes eingetragen. Wird einem Benutzer oder einer Gruppe ein Recht entzogen, so untersucht man bei einem Zugriffsversuch den gesamten Pfad zur Wurzel des Hierarchiebaums, ob dem Benutzer oder der Gruppe das Recht entzogen wurde, um dann den Zugriff zu verweigern.

Das System hat doppelte ACLs und exakt komplementäre GRANT/DENY-Einträge zu verhindern.

Der Zugriff auf die ACLs an sich wird über die Owner-(Besitzer) Eigenschaft der jeweiligen Ressource geregelt. Dadurch wird vermieden, dass auch für die ACLs wieder ACLs eingeführt werden müssen. Die Owner-Eigenschaft wird ebenfalls entlang der Hierarchie vererbt. Dadurch wird erreicht, dass die Administratoren und Manager immer auch die einzelnen ACLs verwalten können. Die Administratoren können weiterhin den Owner einer Ressource ändern, um das System bei dem Wechsel eines Mitarbeiters oder eines Projektes zu einem anderen Mitarbeiter anpassen zu können.

Das Rechtemanagement soll möglichst einfach sein, und es wird, wie in Abbildung 6 erläutert, um ein Rollenkonzept erweitert, das es erlaubt, die Benutzer-System-Schnittstellen an Tätigkeiten anzupassen. Desweiteren ist es die wichtigste sicherheitsrelevante die Außengrenze des Systems.

Die Liste der verwaltbaren Zugriffe ist deswegen minimal gewählt:

- read
- write
- administer

“read” garantiert dem Inhaber das Lesen von Ressourcen, “write” das Schreiben. Das Recht “administer” erlaubt dem Inhaber, die ACL der jeweiligen Ressource zu verwalten. Um Mißbrauch auszuschließen, kann dieses Recht nur vom Owner einer Ressource oder einer Elternressource vergeben werden. Interessant ist ferner, dass auch dem Besitzer einer Kindressource das Recht diese zu administrieren mit Hilfe des Rechts “administer” entzogen werden kann. Dazu muss der Besitzer einer Elternressource dem entsprechendem Benutzer das Recht “administer” durch einen Eintrag in der DENY-Sektion der ACL einer Elternressource entziehen.

Der Aufbau einer ACL erfolgt als einfache Liste von einzelnen GRANT- und DENY-Einträgen. Ein Eintrag enthält folgende Daten:

ACL (element node):

- Principal (User oder Group)
- Zugriff ("read", "write", "administer")
- Start Time
- End Time
- End Date
- Cost

Dabei sind nur die ersten beiden Elemente immer notwendig, nämlich der Benutzer oder die Gruppe und das entsprechende Recht. Alle anderen Elemente sind optional. “StartTime” bezeichnet die Tageszeit, an dem der Eintrag Gültigkeit erlangt, “EndTime” die Tageszeit, an der der Eintrag die Gültigkeit verliert. “EndDate” spezifiziert ein Datum, an dem der Eintrag seine Gültigkeit verliert. Dies kann benutzt werden um ein zeitabhängiges “Copyright” zu implementieren. Das Element “Cost” wird verwendet, um einen kostenpflichtigen Lesezugriff realisieren zu können.

5.3.1.2 Organisationshierarchie

Das vorgeschlagene Rechtemanagementsystem ist streng hierarchisch. Das bedeutet, dass jedes Element und jede Ressource genau ein Elternelement, respektive eine Elternressource hat. Dies hat verschiedene Gründe. Zum einen muss die Rechtevererbung durch geeignete Hierarchie unterstützt werden. Zum anderen soll sich die Realität mit ihren Strukturen und Hierarchien im Rechtemanagementsystem wiederfinden. Dadurch kann die gewünschte und geforderte administrative Autonomie klar definiert und realisiert werden. Neben dem Ziel, dass sich eine Einheit weitestgehend selbst verwaltet und nur in besonderen Fällen hierarchisch höher angesiedelte Administratoren eingreifen können, soll sichergestellt werden, dass sich die Administratoren der höheren Ebenen nur mit der Verwaltung des Systems beschäftigen und keinen direkten Zugang zu den Inhalten haben.

Als einfachst mögliche Hierarchie-Struktur wird ein System definiert, das sich mit einer System-Region als Wurzel rekursiv aus *Regions* zusammensetzt. Als Verwaltungseinheit ist das wichtigste Merkmal jeder Region ihr jeweiliger Besitzer. Jede Region kann autonom Benutzer, Gruppen (und Rollen) und enthaltene Regions verwalten. Um zu strukturieren Systemverwaltung von Systemnutzung zu trennen, wird als weiteres Konzept in die Hierarchie

das *Department* eingeführt. Im Unterschied zum Element Region, das Regions aggregiert, ist ein Department eine organisatorische Einheit, die keine weiteren organisatorischen Einheiten enthält, sondern die konkreten Inhalte, Daten und Verwaltungsinformationen (Benutzer, Gruppen etc.) einer einzelnen Abteilung enthält. Departments sind damit Spezialfälle von Regions, die keine Regions, dafür aber die Resourcen (Medien) der jeweiligen Abteilung enthalten (Abbildung 19).

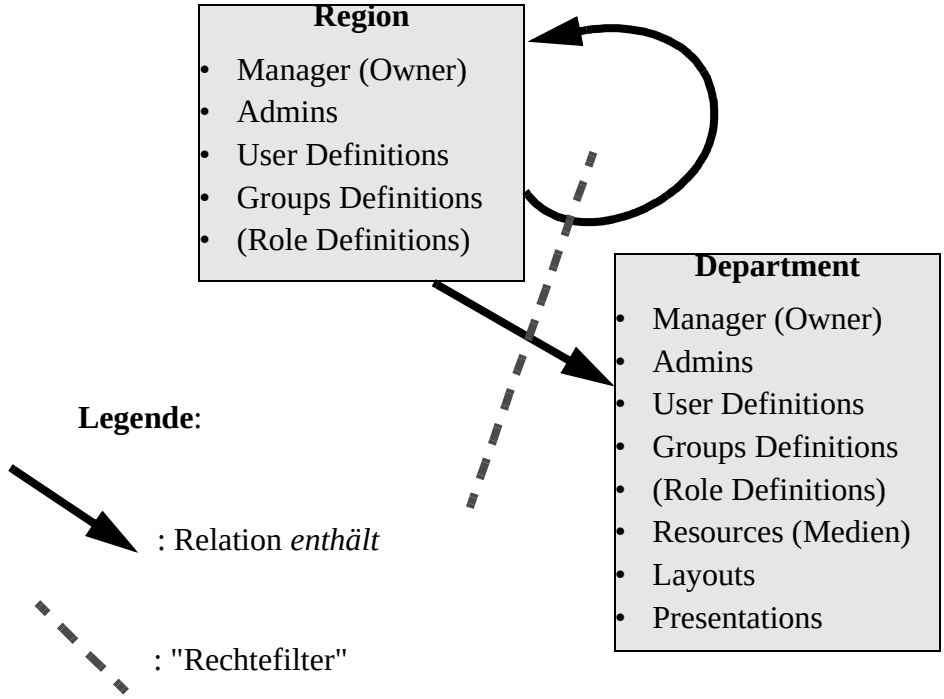

Abbildung 19: Rekursive Organisationsstruktur

Als Sonderfall sind in Abbildung 19 bei den *enthält*-Relationen zwischen Regionen und ihren Sub-Regionen (einschließlich Departments) "Rechtefilter" angedeutet. Diese Rechtefilter bestehen aus DENY-Einträgen an der jeweiligen Subregion, die allen übergeordneten Hierarchien bzw. deren Nutzern jeden Zugriff verbieten. Relevant ist hier insbesondere, dass dies standardmäßig den Administratoren einer Region Einblick und Verwaltung der untergeordneten Regionen verwehrt, sobald diese an ihre neuen Besitzer übergeben werden.

Dazu fügt das Rechtemanagementsystem DENY-Einträge für die systemweite Gruppe "User", die weiter oben in der Hierarchie definiert sind, standardmäßig in die ACL einer neu erzeugten Region (oder eines Departments) ein. Der Besitzer dieser Region ist zunächst, wie bei jedem Element, der erzeugende Benutzer, meistens ein Administrator, so dass er implizit "administration"-Rechte auf dieses Element hat. Sobald er den Besitzer der Region verändert hat, verliert er jedes Recht auf diese Region. Zwei Zusatzoptionen muss das System anbieten: die Möglichkeit für Notfälle, eine Sub-Region wieder "gewaltsam" an sich zu nehmen (immer unter Benachrichtigung des alten Besitzers) und Benachrichtigungen durch das System für Besitzer und Administratoren einer Region, falls der "Rechtefilter" einer Region unvollständig ist (etwa durch Manipulation der ACL vor der Einsetzung des neuen Besitzers).

ie SystemAdmins können unter der MasterRegion verschiedene Regions und/oder Departments erstellen. Danach müssen sie mindestens einen RegionAdmin im Falle einer Region rsp. einen Manager im Falle eines Department einsetzen. Ist dies geschehen, so wird der neu ernannte RegionAdmin oder Manager der Owner (Besitzer) der neuen Ebene, und die SystemAdmins verlieren ihre Rechte auf die von ihnen kreierte Ebene. Gleiches geschieht, wenn ein RegionAdmin eine neue Region oder ein neues Department erstellt. Dadurch wird die Selbstverwaltung weitestgehend sichergestellt. Der Ersteller hat allerdings weiterhin das Recht den RegionAdmin oder den Manager zu wechseln, dies ist aber nur für Notfälle vorgesehen, geht mit einer Systemmeldung einher und ist somit nachvollziehbar.

Der MasterRegion sind die Elemente System, SystemAdmins, Roles, Users und Groups zugeordnet. System ist der System-Benutzer, der in seinen Rechten nicht eingeschränkt ist, er soll nur vom Rechtemanagementsystem selbst benutzt werden. Die SystemAdmins sind sowohl Gruppe als auch Rolle, jedes Mitglied der Gruppe bekommt automatisch die Rolle zugewiesen. Wie alle Admins sind sie für die Verwaltung von Benutzern, Gruppen und Rollen befugt, außerdem können sie Regions und Departments erstellen. Rolleninstanzen und Rollendefinitionen, die von den SystemAdmins erstellt werden, finden sich auch direkt unter der Ebene MasterRegion, ebenso wie alle Benutzer des Medianodesystems. Benutzer können zwar von allen Administratoren auf allen Ebenen erstellt werden, sie gliedern sich jedoch logisch unter der MasterRegion ein, da sie im gesamten System zur Verwaltung benutzt werden können. Daraus folgt allerdings auch, dass die Benutzernamen im gesamten Medianodesystem einmalig sein müssen. Obwohl die Benutzer unter der MasterRegion angeordnet sind, dürfen sie nur von der Admin-Gruppe verwaltet werden, die sie erstellt hat. Das letzte Element unter der MasterRegion sind Systemgruppen, die von den SystemAdmins erstellt und verwaltet werden, aber ebenfalls im gesamten System zur Verwaltung benutzt werden können.

Jeder Region sind die Elemente RegionAdmins, Roles und Groups zugeordnet. RegionAdmins sind ebenfalls Gruppe und Rolle (s.o.), verwalten Benutzer, Gruppen und Rollen und erstellen Regions und Departments. Sie können Rollen definieren oder instanzieren, dadurch sind die Rollen in ihrem Bereich einmalig. Die in einer Region definierten Rollen erhalten als Namen die jeweilige Region vorangestellt. Ebenso wird es mit den Gruppen gehalten, sie werden nach ihrer Heimatregion benannt. Diese Regelung trifft man in ähnlicher Weise bei den AFS-Protection Groups an. Die Ersteller der Gruppen bestimmen über die Mitglieder der Gruppen, die Gruppen selbst hingegen sind für alle sichtbar und können von allen zur Verwaltung genutzt werden.

Jedem Department sind die Elemente Manager, DepartmentAdmins, Roles und Groups zugeordnet. Manager sind nominell die Chefs (Besitzer) ihres Department, beim Einsetzen eines Benutzers als Manager kann er automatisch die damit verbundene Rolle benutzen. Über diese Rolle werden die Rechte des Managers gleich wieder eingeschränkt, um die Fehlbedienung zu vermeiden. Erste Aufgabe des Managers ist es, einen oder mehrere DepartmentAdmins zu ernennen, hier kann er auch sich selbst eintragen. Die DepartmentAdmins sind wiederum gleichzeitig Gruppe und Rolle (s.o.) und verwalten Benutzer, Gruppen und Rollen. Letztere können sie definieren oder instanziieren, die neu definierten Rollen bekommen ebenso

wie die Department-Groups den Department-Namen vorangestellt. Für die Department-Groups gilt das gleiche wie oben für die Region-Groups ausgeführt.

Die Struktur innerhalb der Departments ist in Kapitel 4.3 beschrieben, die für das Rechtemanagement relevanten Elemente sind:

Region (regionname)

- Owner (RegionAdminGroup)
- SystemProperties (Ersteller, Erstelldatum, etc./ nur von System änderbar)
- Properties (Beschreibung, etc./ von Owner änderbar)
- RegionAdmins
- Group
- Role

Department (departmentname)

- Owner (Manager)
- SystemProperties (Ersteller, Erstelldatum, etc./ nur von System änderbar)
- Properties (Beschreibung, etc./ von Owner änderbar)
- Manager
- DepartmentAdmins
- Group
- Role
- Resources
- Layouts
- Presentations

5.3.1.3 Principals: Gruppen

Alle ACLs im System verbinden Principals, das sind Gruppen und Benutzer, mit Inhaltsdaten des Systems (Medien, Metadaten, System- und Benutzerdaten). Gruppen und Benutzer sind jeweils eindeutig innerhalb einer Verwaltungseinheit (Region) definiert.

Notwendig sind zum einen ACLs, die für alle Benutzer gelten ("All"), insbesondere für den Rechtefilter der Region-Hierarchie. Zum anderen muss die Organisation der Regionen ihre Entsprechung bei der Benutzerverwaltung finden können.

Als Resultat sind die Gruppen - *Group*s - streng hierarchisch organisiert:

- eine Gruppe ist entweder die vordefinierte Gruppe "All" der MasterRegion oder
- als Untergruppe genau einer anderen Gruppe (etwa "All" von MasterRegion) definiert, die keine (auch mittelbare) Untergruppe dieser Gruppe ist

Das System darf nur Änderungen an Gruppendefinitionen erlauben, die diese Regel nicht verletzen. Die Änderung und die Spezialisierung von Gruppen durch neue Gruppen ist mit den "administer"-Rechten der entsprechenden Region (respektive Departments) verbunden. Dies erfordert standardmäßig eine lokale Gruppe "All" für jede Region vor der Übergabe einer

Region vom Erzeuger an ihren Manager, damit der Manager autark Gruppen und Benutzer einführen und verwalten kann.

Für jede Region ist eine Gruppe *Admins* sinnvoll, für die MasterRegion ist eine solche Gruppe definiert.

ACLs, die Festlegungen für eine Gruppe machen, gelten auch für alle Untergruppen. Bei kollidierenden ACLs an einem einzelnen Datenelement gilt jeweils die Angabe für die speziellste enthaltende oder identische Gruppe.

Group (regionname/departmentname:groupname)

- Owner (Erstellergruppe [Admins])
- SystemProperties (Ersteller, Erstelldatum, etc./ nur von System änderbar)
- Properties (Beschreibung, etc./ von Owner änderbar)

5.3.1.4 Principals: Benutzer

Einzelne Benutzer werden von Region-(Department-) Administratoren als *User* erstellt. Jeder Benutzer wird bei Erstellung einer Gruppe zugeteilt. Das Zuordnen eines Benutzers zu einer Gruppe erfordert sowohl für den Benutzer wie für die Gruppe Schreibrechte, was nur die jeweils lokale Administratorengruppe standardmäßig hat. Insbesondere bedeutet das regelmäßig, dass innerhalb einer Region keine Benutzer Gruppen übergeordneter Regionen zugeordnet werden können.

User (username)

- Owner (Erstellergruppe [Admins])
- SystemProperties (Ersteller, Erstelldatum, etc./ nur von System änderbar)
- Properties (Beschreibung, etc./ von Owner änderbar)
- Profile (bevorzugte Sprache, bevorzugte Bildschirmauflösung, etc./ vom Benutzer änderbar)

5.3.1.5 Administrierte Elemente, Resourcen

Alle komplexen Elemente der Datenstruktur lassen sich auf eine Grundstruktur zurückführen: jedes Element besteht aus einer Ressource und einer mit ihr assoziierten ACL, sowie abhängig von der Art des Elements aus optionalen Kindressourcen. Jede Hauptresource oder jedes Element verfügt über eine Owner-Kindresource, in der der jeweilige Besitzer oder die Besitzergruppe eingetragen werden. Jede Ressource vererbt ihre ACL an ihre Kindresourcen - eine Ausnahme bildet hier die Owner-Kindresource, die lediglich den Besitzer und die zugehörige Administratorgruppe beinhaltet.

Alle Elemente sowie Ressourcen des vorgeschlagenen Systems werden bei ihrer Erstellung mit einer systemweit eindeutigen ID versehen. Dazu wird ein Algorithmus zur Anwendung kommen, der die Einmaligkeit der generierten ID mit hoher Wahrscheinlichkeit sicherstellen kann. Die bei der Erstellung vergebene ID begleitet das Element oder die Ressource bis sie aus dem System gelöscht wird. Sie überdauert auch das Umbenennen des Elements oder der Ressource. Dieser Ansatz wird benutzt, um jedes Element und jede Ressource stets eindeutig identifizieren zu können.

Jedem administrierbaren Element, also jedem Element, das eine ACL hat, ist ein Benutzer als Besitzer, *Owner,* zugeordnet. Administrierbare Elemente sind Regions, Departments, Ressourcen und ResourceTrees, Layouts und layoutTrees, Presentations und PresentationTrees (vrgl. Kapitel 4.3). Für all diese Elemente

Element (elementname)

- Owner (Erstellergruppe [Admins])
- ACL
- SystemProperties (Ersteller, Erstelldatum, etc./ nur von System änderbar)
- Properties und Inhalt (Beschreibung, etc./ von Owner oder wg. ACL änderbar)
  Language
  Texttype
  Textsize
  Textcolor
  etc.,
  evtl. Inhalt (bei Monomedien) oder Unterresourcen

5.3.2 Systemvorgaben

Systemvorgaben existieren, um die Funktionsfähigkeit und Handhabbarkeit im vorgeschlagenen System zu gewährleisten. Insbesondere standardmäßige Rollen- und Gruppenvorgaben werden definiert.

Weiterhin ergaben sich aus den diskutierten Szenarios folgende Vorgaben für den Betrieb eines Vorlesungsarchivs:

- SystemProperties werden nur vom System vergeben,
- Admins erstellen und verwalten Benutzer, Gruppen und Rollen,
- Admins haben grundsätzlich keine Rechte auf Ressourcen,
- SystemAdmins und RegionAdmins erstellen zusätzlich Regions und Departments,
- Gruppen, Rollen, Departments und Regions dürfen nur gelöscht werden, wenn sie leer sind,
- Erstellt ein SystemAdmin einen Benutzer, eine Gruppe oder eine Rolle, so wird als Besitzer die SystemAdminGroup eingetragen,
- Erstellt ein RegionAdmin einen Benutzer, eine Gruppe oder eine Rolle, so wird als Besitzer die regionname:RegionAdminGroup eingetragen,
- Erstellt ein DepartmentAdmin einen Benutzer, eine Gruppe oder eine Rolle, so wird als Besitzer die departmentname:DepartmentAdminGroup eingetragen,
- Admins dürfen nur die Eigenschaften der Benutzer, Gruppen und Rollen ändern, die ihre entsprechende Gruppe angelegt hat,
- Über die Mitglieder von Gruppen bestimmt die Administratorengruppe, die die Gruppe besitzt, benutzen darf die Gruppe hingegen jeder Benutzer,
- Der Manager ernennt seine DepartmentAdmins, er darf sich auch selbst als DepartmentAdmin eintragen,
- SystemAdmins und RegionAdmins dürfen die RegionAdmins der von ihnen kreierten Regions notfalls ändern, dies ist mit einer Nachricht (s.u.) verbunden,

- SystemAdmins und RegionAdmins dürfen die Manager der von ihnen kreierten Departments notfalls ändern, dies ist mit einer Nachricht (s.u.) verbunden,
- SystemAdmins und RegionAdmins dürfen die Verwaltung von Benutzern und Gruppen ihrer Kindebenen übernehmen (den Besitz übernehmen), dies ist mit einer Nachricht verbunden,
- DepartmentAdmins dürfen den Besitzer von Ressourcen in ihrem Department ändern (anderer Benutzer oder Gruppe), dies ist mit einer Nachricht verbunden,
- Der Besitzer einer resource darf alle ACLs der Kindresourcen seiner Ressource bearbeiten, dies ist mit einer Nachricht verbunden.

5.3.3 Benachrichtigungen

Nachrichten dienen vornehmlich dazu, möglichen Mißbrauch des Systems durch besonders privilegierte Benutzer aufzudecken und dadurch zu unterbinden. Die für die erste Implementierungsstufe vorgesehenen Nachrichten sind im folgenden aufgeführt:

Ändert ein SystemAdmin oder ein RegionAdmin die Mitglieder der RegionAdminGroup einer von ihm erstellten Region, so erfolgt eine Nachricht an die betroffene RegionAdminGroup, an alle anderen RegionAdminGroups dieses Astes die Hierarchie hinauf, sowie an die SystemAdminGroup.

Ändert ein SystemAdmin oder ein RegionAdmin den Manager eines von ihm erstellten Departments, so erfolgt eine Nachricht an den betroffene Manager sowie die betroffene DepartmentAdminGroup, an alle anderen RegionAdminGroups dieses Astes die Hierarchie hinauf, sowie an die SystemAdminGroup.

Übernimmt ein SystemAdmin oder RegionAdmin die Verwaltung eines Benutzers oder einer Gruppe, die von einem RegionAdmin oder einem DepartmentAdmin erstellt wurde, so erfolgt eine Nachricht an den betroffenen Benutzer, rsp. die Gruppe, die entsprechende RegionAdminGroup rsp. DepartmentAdminGroup sowie die SystemAdminGroup.

Wenn ein DepartmentAdmin den Besitz einer Ressource jemand anderem zuweist, werden Nachrichten an den bisherigen Besitzer (eventuell Vetorecht), die jeweilige DepartmentAdminGroup sowie den Manager des Department gesendet.

Wird von einem Admin eine Rollendefinition bearbeitet, erstellt oder gelöscht, die instanziiert werden kann, so erfolgt eine Benachrichtigung der hierarchisch betroffenen RegionAdminGroup und DepartmentAdminGroup.

Ändert ein Besitzer die ACL einer Kindresource, deren Besitzer er nicht ist, so erfolgt eine Meldung an den Besitzer der Kindresource.

5.3.4 Spezifizierung des Rechtekonzeptes

Zugriffsrechte

Der Datenbestand ist eine Hierarchie von Knoten, deren Blätter auch Attributknoten sein können:

$N = N_{\text{elements}} \cup N_{\text{attributes}}$ ist die Menge aller Knoten mit $N_{\text{elements}} \cap N_{\text{attributes}} = \varnothing$.

$E \subset N_{\text{elements}} \times N$ ist die Menge aller gerichteten Kanten, wobei

(N, E) die Hierarchie des Datenbestandes darstellt. Die Wurzel sei $n_{\text{root}} \in N_{\text{elements}}$, die Funktion parent: $N \to N_{\text{elements}}$ die Umkehrung der Kantenrelation. Die Funktion $\text{Ancestor}(n) = (\{n\} \cap N_{elements}) \cup \{n_a | (n_a, n) \in E \vee \exists (n_b, (n_a, n_b) \in E \wedge (n_b, n) \in E)\}$ definiert damit den Pfad über die Element-Knoten von n zur Wurzel n_{root}.

An den Elementknoten Region und Departement können Principals, Benutzer oder Gruppen, definiert werden: $P_n = P^u{}_n \cap P^g{}_n$. Ein Principal ist an einem Knoten definiert, wenn er an einem Knoten aus Ancestor(n) definiert ist: $p(n) = p^u(n) \cap p^g(n)$ mit $p^u(n) = \bigcup_{a \in \text{Ancestor}(n)} P^u{}_a$, entsprechend $p^g(n) = \bigcup_{a \in \text{Ancestor}(n)} P^g{}_a$.

Diese Knoten sind mit ACLs annotiert: $\text{acl}_n \in (P, G, A)^*$, wobei

$G = \{\text{grant, deny}\}$ Gewährung oder Entzug eines Rechtes bedeuten soll

$A = \{\text{read, write, administer, changeowner, delete, createchild, deletechild}\}$ mögliche Aktionen im Datenbestand, und mit $(p, g, a) \in \text{acl}_n \Rightarrow p \in p(n)$ nur am Knoten definierte Principals zulässig sind.

Die Menge der möglichen Zugriffe auf einen Knoten n sei durch folgende Funktion definiert:

$\text{access}: N \to (P, A), \text{access}(n) = \{(p, a) | (p, \text{grant}, a) \in \text{acl}(n) \wedge \neg((p, \text{deny}, a) \in \text{acl}(n))\}$

mit $\text{acl}(n) = \bigcup_{a \in \text{ancestor}(n)} \text{acl}_a$ als Menge aller am Knoten "bekannten" ACLs. (1)

5.3.5 Realisierung urheberrechtlicher Aspekte

Ein Rechtemanagementsystem muss also folgende Punkte, die sich aus den oben rudimentär beschriebenen rechtlichen Anforderungen ergeben, abbilden können:

a. Recht zur kostenpflichtigen Nutzung,
b. Recht zur Beschränkung der Zahl der Zugriffe auf eine Ressource,
c. Recht zur Beschränkung der Zahl gleichzeitiger Zugriffe auf eine Ressource,
d. Identifikation von Ersteller und Arbeitgeber zum Erstellzeitpunkt jederzeit,
e. Historie, Nachvollziehbarkeit der Veräußerung von Zugriffsrechten,
f. Möglichkeit zur Verschiebung von Ressourcen (Verkauf),
g. Mehrfachbesitzmöglichkeit,
h. Ablage von und Verweismöglichkeiten auf Verträge.

Diese abgeleiteten Anforderungen lassen sich nun wie folgt im vorgestellten System darstellen.

Um die Anforderungen an den Zugriff, bzw. die Nutzung von Ressourcen zu erfüllen, müssen die ACL-Rechte angepaßt werden. Dazu könnten die Rechte-Einträge für Benutzer und Gruppen erweitert werden zu:

- Right
- StartTime
- EndTime
- EndDate

- Cost
- Credit
- Concurrent

Neu hinzugekommenen sind die optionalen Elemente “Credit” und “Concurrent”. Das “Cost”-Element wurde bereits im vorangegangenen Kapitel eingeführt, es ermöglicht die kostenpflichtige Wahrnehmung von Rechten (Punkt a.). “Credit” wird benutzt, um die absolute Zahl der Zugriffe auf eine Ressource zu begrenzen. Dazu wird “Credit” bei jedem Zugriff um eins vermindert. Erreicht “Credit” den Wert Null, so ist das Recht zu verweigern (Punkt c.). “Concurrent” bezeichnet die Zahl der maximal gleichzeitig zugreifenden Benutzer und macht in diesem Zusammenhang nur in Verbindung mit einer Benutzergruppe Sinn (Punkt b.).

Um nachvollziehen zu können, wer eine Ressource ursprünglich kreiert hat, ist es notwendig, dass der Ersteller einer neuen Ressource und sein derzeitiger Arbeitgeber für immer mit der Ressource verbunden bleiben. Dies ist auf einfache Art und Weise zu realisieren, indem die jeweilige ID des Benutzers und des Department in der die Ressource erstellt wurde vom System in den SystemProperties abgelegt werden (Punkt d.). Außerdem muss in einem geeigneten Element der Verkauf einer solchen Ressource nachvollziehbar sein. Dazu müßte in diesem Element ein Eintrag pro neuem Department und entsprechendem Datum erfolgen (Punkt e.).

Ist eine Gruppe statt eines einzelnen Benutzers für die Erstellung einer Ressource verantwortlich, so kann bei der Erstellung einer Ressource ein Vertrag mit der Ressource verknüpft werden, der später nicht mehr gelöscht werden kann. In diesem Vertragselement wird dann festgelegt werden, wer alles Urheber der neu erstellten Ressource ist, auch wenn nur eine Benutzer-ID in den SystemProperties vermerkt ist. Eine andere Möglichkeit dieses Problem zu lösen, besteht darin, dass bei der Erstellung einer Ressource vom System nach den Erstellern gefragt wird, diese dann angegeben und nachträglich nicht mehr geändert werden können. Als Besitzer der neuen Ressourcen könnten dann optional die genannten Benutzer übernommen werden. Eine andere Möglichkeit besteht darin, dass der entsprechende DepartmentAdmin die Personen von Hand als Besitzer der Ressource hinzufügt (Punkt g.).

Betrachtet man nun den Fall, dass ein Manager sämtliche Nutzungsrechte an einer Ressource oder einem Ressourcenbaum verkauft, dann werden diese Ressourcen aus dem Department des Managers in ein anderes Department verschoben. In diesem Fall muss die Funktionalität der Hierarchie in dem Ursprungsdepartment sichergestellt bleiben. Dieses kann man erreichen, indem man “verkaufte” Ressourcen nach deren Verschiebung nicht löscht, sondern sie durch leere Ressourcen ersetzt, deren einziger Zweck die Aufrechterhaltung der Struktur ist (Punkt f.).

Wird eine der oben genannten Ressourcen verschoben, so behält sie natürlich ihre ID, so dass die Verträge, die in einem Department nach dem Verkauf verbleiben immer zugeordnet werden können. Der eigentliche Verkaufsvertrag wird sowohl in dem Department des verkaufenden Managers als auch in dem Department des kaufenden Managers abgelegt.

Kapitel 6 - Nutzer- und Autoren-Rollen

Multimediale Inhalte sind komplex strukturiert, auch der Vorgang und die Organisation der Erstellung ist komplex. Ab einem gewissen Aufwand muss die Erstellung eines einzelnen Projektes auf mehrere Autoren und in parallele Arbeitsvorgänge aufgeteilt werden können. Dabei kann die konsistente (Re-) Kombination der Arbeitsergebnisse durch eine Fokussierung der Möglichkeiten der Autoren in ihren jeweiligen Arbeitsvorgängen gesteuert und vereinfacht werden.

In diesem Kapitel wird hierzu ein generisches Rollenkonzept für Arbeitsvorgänge erarbeitet. Dieses Rollenkonzept ergänzt das Rechtemanagement. Es soll keine harte Sicherheit bieten, wie sie etwa für die Wahrung von Urheberrechten notwendig ist. Vielmehr soll das Rollenkonzept dem Nutzer bzw. Autor, der eine Arbeit mit bestimmtem Fokus vornimmt, helfen. Rollen bieten Sicht und Zugriffsmöglichkeiten auf das System, die auf die jeweils spezielle Arbeit fokussieren und damit Vorgänge einfacher und auch sicherer machen.

Durch Rollen sollen Rechteanhäufungen, wie sie in einem System mit Personalunionen auftreten, aufgelöst werden können. Dadurch werden Fehler vermieden, etwa beim Administrator, der aus Versehen Inhalte verändert, statt sie zu verwalten. In einem System mit Rollen mussmuss er sich dazu in zwei verschiedenen Rollen anmelden. Rollen werden im folgenden, insbesondere in Abschnitt 6.2, klar von Gruppen differenziert.

Rollen-Information soll nicht nur verwendet werden können, um verschiedene grafische Sichten (*GUI*s) zu erzeugen, sondern auch in anderem Kontext maschinell benutzt werden, um Datentransformationen auf etwaige Risiken zu kontrollieren.

6.1 Motivation

An den drei Beispielen aus Abschnitt 2.1 soll die Notwendigkeit der Unterstützung von festgelegten Arbeiten durch definierte Rollen belegt werden.

Unter dem Gesichtspunkt Nutzer- und Autorenrollen als komplexeste Anwendung wird im Abschnitt Kapitel 6.1.3 die Anwendung aus Abschnitt 2.1.3, ein verteiltes Vorlesungsarchiv, diskutiert. Für diese Anwendung wird im weiteren Fortgang dieses Kapitels ein Rollenmodell entworfen, das eine Verallgemeinerung der notwendigen Rollenmodelle der ersten beiden Beispiele darstellt.

6.1.1 Beispiel I: Kiosksystem

Das interaktive, multimediale Kiosksystem aus [51] gliedert seine Autoren in drei Gruppen: Applikationsdesigner, Autoren des Corporate Design und Inhaltsautoren, daneben noch Benutzer (“Null-Autoren”) und Programmierer, welche dem System als Autoren der Bezüge zu externer Semantik erscheinen.

Auch feinere Abstufungen zwischen diesen Basisrollen können nützlich sein, sind aber ohne weitere Hilfsmittel nur durch hohen Aufwand, etwa Erweiterung der Datenstruktur und entsprechende Implementierungen im Präsentationsprogramm, möglich.

[51] beschreibt einen Weg, wie die vom System benötigten Rollen durch ein einziges Präsentations-Programm realisiert werden können. Die Rollen werden hierbei implizit und für die ganze Anwendung in der Implementierung des Programmes definiert. Das Datenschema der Anwendung jedoch bildet sehr explizit die Semantik eines multimedialen Kiosks ab, was dem System die nötige Information liefert um die Rollen zu realisieren. So ist etwa die Unterscheidung zwischen Layout (hier: “Templates”) und Inhalt explizit im Datenbestand vorhanden, um die Tätigkeiten des Designers und des Inhaltsautoren trennen zu können.

Dieses Modell hat sich in großen, aber organisatorisch zentralistischen Applikationen bewährt. Hier werden Rollen und Zuständigkeiten zentral einheitlich und normativ vorgegeben. Auch die hohen Kosten dafür, für die vorgegebene Menge an Arbeitsvorgängen die Rollen in ein neues Präsentationsprogramm zu kodieren, werden durch die hohe Anzahl von genutzten identischen Kopien des Programms relativiert. Zudem ist die oben beschriebene Rollenverteilung in dieser Klasse von Anwendungen überwiegend einheitlich anzutreffen. Der Grad der Anpassung des Rollenmodells an die jeweiligen Bedürfnisse des Kunden konkurriert mit dem Anpassungsaufwand und schränkt in diesem Modell die Anzahl der Rollen auf nur wenige, festgelegte ein.

6.1.2 Beispiel II: News-On-Demand-System

Das in Abschnitt 2.1.2 kurz vorgestellte News-On-Demand-System HyNoDe findet folgende Menge an Arbeitschritten und damit durch entsprechende Werkzeuge zu unterstützende Autorenrollen (vrgl. Abbildung 1 auf Seite 6).

- Redakteure annotieren Monomedien und bilden einzelne Präsentationen (“Stories”) durch Zuordnung von Layouts zu einer Menge von Monomedien. Die einzelnen Stories sind zum einen um vieles kleiner als eine typische Kioskanwendung, zum anderen werden sie aus bereits vorhandenem monomedialen Material zusammengesetzt.
  Dies unterscheidet sich stark von der nur grob entsprechenden Rolle des Applikationsdesigners eines Kiosksystems. Dieser schafft die komplexen Grundlagen für eine zusammenhängende, große Anwendung, ohne die darin zu integrierenden Monomedien vorher zu kennen.
- Die Rollen der Designer sind recht ähnlich, wobei die Layouts für ein News-On-Demand-System typischerweise deutlich weniger Kriterien genügen müssen als die für ein Kiosksystem. In ersterem reicht oft die Beschreibung der (Höchst-) Anzahl und Art der beteiligten Monomedien für die zuverlässige Erstellung der (multimedial) einfach strukturierten Stories. In letzterem ist die vorhandene, oft sehr große Menge an Monomedien zu einer einzigen Anwendung verbunden. Oft sind durch die enge Interaktion mit externen Programmen viel genauere formale Vorgaben gegeben, etwa über Ausnutzung von Bildschirmflächen.

- Inhaltsautoren, hier Journalisten, arbeiten in HyNoDe ausschließlich mit Monomedien, denen sie nur Angaben über die Herkunft mitgeben. Bei einem Kiosksystem müssen meistens Inhalte innerhalb eines schon strukturierten Systems integriert werden, wenn auch auf uniforme Art (etwa durch simples Anhängen zusätzlicher Präsentationsseiten).

Die resultierende Rollenverteilung unterscheidet sich nicht nur in der Arbeitsverteilung auf die einzelnen Rollen der Rollenverteilung des Kiosk-Systems im Abschnitt 6.1.1. Ein wichtiger Unterschied besteht auch in der komplexeren Organisation der beteiligten Autoren.

6.1.3 Beispiel III: Vorlesungsarchiv

In einem Archiv zur gemeinsamen Nutzung von multimedialem Vorlesungsmaterial gibt es auch den Bedarf zur Unterstützung verschiedener Arbeitsschritte, ausgeführt von verschiedenen Rollen. Die Rollen, die hier benötigt werden, sind wie in den beiden vorigen Beispielen Inhalts- und Designautoren. Weiterhin ergeben sich aber durch die aufwändige systemweite Organisation der beteiligten Autoren, des Inhalts und durch die notwendige dynamische/lokale Verwaltung von Rollen weitere notwendige, u.a. administrative, Rollen.

Die Anforderungen sind in folgendem komplexer als bei den beiden vorigen Applikationen:

- Es existieren viele verschiedene organisatorische Einheiten, denen Autoren und Inhalte zugeordnet werden müssen, wie für einen News-On-Demand-Service mit mehreren Inhaltsanbietern. Im Gegensatz dazu aber müssen Benutzer systemweit bekannt sein, da sie sich gezielt Material zur Verfügung stellen können sollen.
- Noch gravierender ist ein Unterschied, der direkt die Rollen betrifft: für diese Applikation gibt es keine Zentrale wie für einen Konzernkiosk, die Arbeitsschritte und Zuständigkeiten vorschreiben kann, sondern einzelne, sehr autonome und typischerweise sehr autokratische, individuelle Lehrstühle. Auch sind die Rollen nicht so klar durch vorhandene Berufsbilder wie bei einem News-On-Demand-System gegeben, da Lehrstühle sehr verschiedene Arten von Organisation, Diversifizierung der Tätigkeiten und Zuständigkeiten aufweisen.

Die in Abschnitt 4.3 gewählte Datenstruktur stellt für die erste Bedingung eine ausreichende Lösung dar. Die zweite Charakteristik dieser Anwendung jedoch schließt die alleinige und feste Implementierung systemweiter Rollen aus. Um auch hier eine zum Beispiel rein inhaltsbezogene Arbeit von Autoren zu ermöglichen, und vom System zu unterstützen, dass diese Arbeit nicht mit formalen oder anderen lehrstuhlweiten Vorgaben kollidiert, müssen solche Vorgaben dynamisch und lokal ins System gebracht werden können.

Wegen der dezentralen, aber verbundenen Organisation der Autoren ergab die Analyse der Anforderungen eines verteilten Vorlesungsarchivs im Rahmen des Projekts Medianode unter anderem die folgende Anforderung: jeder Lehrstuhl benötigt eine einzige, zentrale Stelle, die nach Vorgaben des Lehrstuhlinhabers Rollen definiert. Diese Funktion *RoleAdmin*, wird vorerst bei den DepartmentAdmins lokalisiert (ss.. Abschnitt 4.3) - sie muss, unter Benutzung des unten gegeben Rollenmechanismus, auch separiert oder anderen Arbeiten zugeordnet werden können. Ein RoleAdmin braucht die Möglichkeit eigene, neue Rollen zu erstellen oder vorgegebene Rollendefinitionen (übergeordneter Administratoren) zu benutzen. Rollen können von

berechtigten Administratoren als möglichen Rollen Benutzern (innerhalb des von ihnen verwalteten Lehrstuhls) zugewiesen werden.

Die hier addressierten Nutzer-Rollen sind nicht die Rollen für Typen oder Klassen in (auch objekt-orientierten) Datenbanksystemen: das sind die verschieden Relationen, Nutzungen, in denen Typen oder Klassen von Daten im Schema einer Datenbank verwendet werden. Diese Rollen von Klassen innerhalb eines Typschemas lassen sich jedoch als Information über die Semantik eines Datenbestandes verwenden: wo z.B. bekannt ist, dass ein Datum der Klasse *TextDokument* in der (Klassen-) Rolle *Layout* verwendet wird, könnte etwa der ändernde Zugriff darauf nur Designern angeboten werden.

6.2 Anforderungen an ein Rollenkonzept

Das dezentral realisierte Archiv zur kooperativen Nutzung und Pflege von multimedialen Vorlesungsinhalten aus Abschnitt 2.1.3 soll als im Hinblick auf die Nutzungsstruktur komplexestes der drei in Kapitel 2 vorgestellten Systeme im weiteren Grundlage für den Entwurf eines Rollenkonzepts sein. Im Rahmen des entsprechenden Projektes Medianode erwies sich eine generische Unterstützung der verschiedenen Arbeitsschritte als noch ungelöst.

Ein Charakteristikum einer Anwendung wie in Abschnitt 2.1.3 ist es, dass eine Person verschiedene Aufgaben im Medianodesystem ausüben wird. Dies ergibt sich aus typischer Größe und Zusammensetzung der Teams an hiesigen Lehrstühlen, wo selten der Administrator nur administriert und der Autor nur schreibt und redigiert. Es ist vielmehr so, dass es vielfältige Personalunionen gibt. Das bedeutet, dass eine reale Person verschiedene Rollen ausübt. Diesem Umstand kann man auf verschiedene Arten begegnen. Im Windows 2000-System trifft man die wohl umfangreichste Implementierung der vorgestellten existierenden Systeme an (Abschnitt 3.2). Hier wird mit vielfältigen Systemgruppen gearbeitet, die eine Rollenfunktion für die Mitglieder übernehmen. Allerdings werden, wie bereits im vorangegangenen Kapitel ausführlich besprochen, die Rechte eines Benutzers aus seinen Gruppen addiert, so dass ein Benutzer, der sowohl Administrator als auch Autor ist, stets mit (mindestens) Administratorrechten im System arbeitet, auch wenn er nur einen Text editieren möchte. Die Lösung, die im Windows-System vorgeschlagen wird, besteht darin, dass für jede Rolle, die ein Benutzer innehat, ein neuer Windows-Benutzer angelegt wird. Dies ist aber eine fehleranfällige und mit hohem Administrationsaufwand verbundene Lösung. Im vorgeschlagenen System hingegen wird mit einem neuen Element, nämlich der Rolle gearbeitet. Über Rollen ist es möglich, die Rechte, über die ein Benutzer aus der Addition seiner Einzel- und Gruppenrechte verfügt, auf ein Maß, welches seiner aktuellen Rolle genügt, zu vermindern. Der Vorteil eines solchen Systems liegt zum einen darin, dass pro realem Benutzer auch nur ein Benutzer im System gepflegt werden muss und sich dieser auch nur ein Paßwort merken muss. Zum anderen darin, dass über das Rollensystem auch Strukturen, die im System immer wieder auftreten, für bestimmte Rollen verwaltet werden können. So ist es dann z.B. möglich, der Rolle “Kosmetiker” lediglich das Recht einzuräumen, die Textfarbe aller Ressourcen im System zu bearbeiten. Da, wie oben dargestellt, eine systemweit einheitliche, feste Definition aller Rollen nicht zu

einer dezentralen Organisation der Nutzer passt, wird eine generische Definitionsmöglichkeit solcher Rollen vorgestellt.

Rollen werden hier unter dem Gesichtspunkt der Koordination vieler Autoren betrachtet, für eine Anwendung, die Lehrstühle unterstützen soll, sind sie zuerst auf Lehrstuhlebene notwendig. Rollen können sehr wohl auch von Individuen benutzt werden, um sich selbst zu koordinieren. Das unten vorgestellte Konzept zur Spezifizierung und Semantik von Rollen ist auch dafür direkt verwendbar. Da die Administration von Rollen nicht trivial ist, wird sie im Rahmen der Arbeitsteilung lediglich den Administratoren obliegen. Für ein vernünftig verwaltbares und genügend übersichtliches System ergab sich im Rahmen von Medianode eine Konzentration der Rollendefinition auf eine Stelle und die entsprechend autorisierten RoleAdmins pro Lehrstuhl als opportun.

Die erste Anforderung an eine Rolle ist die Unterstützung des Arbeitsschrittes, für den diese Rolle kreiert wird. Dazu muss genügend Information der bearbeiteten Daten im System in bezug zur Semantik dieses Arbeitsschrittes gesetzt werden können. Um für Rollendefinitionen eine systemweit einheitliche Semantik anbieten zu können, müssen klare Kriterien vorhanden sein, was zur Rollendefinition verwendet werden kann.

Es gibt Informationen und Arbeitsschritte, die in der Erstellung und Pflege des Inhalts jeder multimedialen Applikation vorkommen werden (etwa Besitzrechte und deren Administration). Es gibt aber auch - zunehmend anwendungsspezifische - Informationen und Arbeitsschritte, die nur für Medieninhalte eines bestimmten Kontextes sinnvoll sind. Die gewählte Zielapplikation jedoch legt die Medieninhalte nicht thematisch fest, und somit kann auch der Datentyp nicht weiter auf bestimmte Inhalte spezialisiert werden, etwa durch entsprechend spezifische Metadaten. Wäre das der Fall, könnte das zu entwickelnde Rollenkonzept diese themenspezifischen Informationen verwenden. Auf die Wahl des Abstraktionsgrads des Rollenkonzepts geht weiter unten Abschnitt 6.4 ein.

Eine weitere Anforderung an ein Rollenkonzept ist, dass es sich klar und sinnvoll von den Zugriffsrechten, insbesondere der Definition von Benutzergruppen abhebt. Ist das nicht der Fall, ist der praktische Sinn eines eigenen Rollenkonzeptes sicher infrage gestellt.

Die Applikation sollte erkennbar zwischen Zugriffsrechten als Mittel zur Sicherstellung von Urheberschaft, Konsistenz und Nutzungsrechten verschiedener Nutzer und Rollen als Unterstützung verschiedener Arbeiten (auch eines einzelnen Nutzers) durch Fokussierung seiner Zugriffsmöglichkeiten unterscheiden.

6.3 Definition des Rollenkonzepts

Zur Realisierung von Rollen gibt es generische Eigenschaften, die in der Datenstruktur aus Kapitel 4.3 vorhanden sind und die direkt in Relation mit vorkommenden Arbeitsschritten gesetzt werden können:

- Trennung von Layout und Inhalt
  Dies erlaubt dem System zu erkennen, ob inhaltliche Arbeit oder redaktionelle Änderungen (Verbindung von Resourcen und Layout zu Präsentationen) gemacht werden,
- Explizite Metadaten
  Metadaten sind in Form explizit benannter Relationen und Attribute gegeben. Diese Attribute und Relationen können oft bestimmten Arbeiten zugeordnet werden. Als Beispiel kann die Schriftgröße oder Farbe von Text dienen (Design), die Strukturierung des Inhalts zu alternativen/semantisch äquivalenten oder auch aggregierten Inhalten (Inhaltliche Arbeit) und etwa die annotierten Rechte-Informationen (Administration).

Im Gegensatz dazu sind auf den konkreten Inhalt bezogene Angaben zu Autorenrollen nicht klar von der normalen Rechte- und Zugriffsverwaltung zu trennen und sind vor allem nur themenbezogen, nicht aber themenunabhängig verwendbar.

Unter anderem aus diesen Gründen wird das vorgestellte Rollenkonzept keine konkreten System- und Medieninhalte (außer der reinen Rollendefinitionen selbst) auswerten (vergleiche auch Abschnitt 6.4).

6.3.1 Rollen

Die Rollen sollen dazu dienen, den durch das Rechtemanagement bestimmten Zugriff weiter einzuschränken, um mit einer verkleinerten Menge von Aktionen und Aktionsobjekten die Bedienung einfacher und fehlerresistenter zu gestalten. Rechte-ACLs sind immer mit einer Ressource rsp. einem Element verbunden. Dies ist bei Rollen nicht der Fall: ihre Einschränkungen gelten systemweit. Um dies zu ermöglichen, wird nicht der dynamisch veränderliche Inhalt, sondern das statische Typschema des gesamten Inhalts mit Rolleninformation annotiert (Abbildung 21). Die Definition der Einschränkungen durch Rollen werden ausschließlich von den Administratoren vorgenommen und verwaltet - der “normale” Benutzer kommt mit dieser Verwaltung nicht in Kontakt. Der Vorteil dieses zusätzlichen Konzepts liegt in den immer wiederkehrenden Strukturen im Datenbankschema, die durch einfache Definition im gesamten Medianodesystem zur Rechteverwaltung genutzt werden können. So ist zum Beispiel denkbar, dass ein Benutzer in einer bestimmten Rolle nicht das Recht haben soll, die Schriftgröße von Ressourcen jeder Art zu ändern. Dies erfolgt dann im Datenbankschema durch die Beschränkung der Verfolgung der Relation zwischen einer Ressource und dem ihr zugeordneten Element Schriftgröße. Diese Beschränkung hat dann im gesamten Medianodesystem Gültigkeit

und ist damit nicht mehr, wie die ACLs, von der zugehörigen Ressource oder der Hierarchie abhängig.

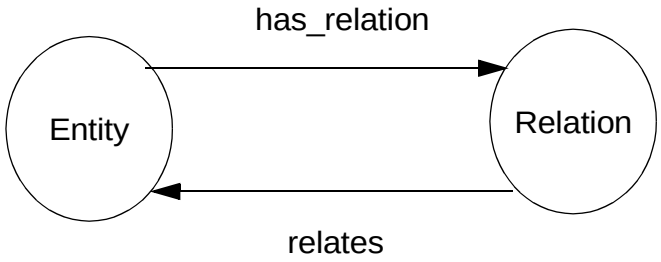

Abbildung 20: Meta-Schema einer relationalen Datenbank (Entity-Relationship)

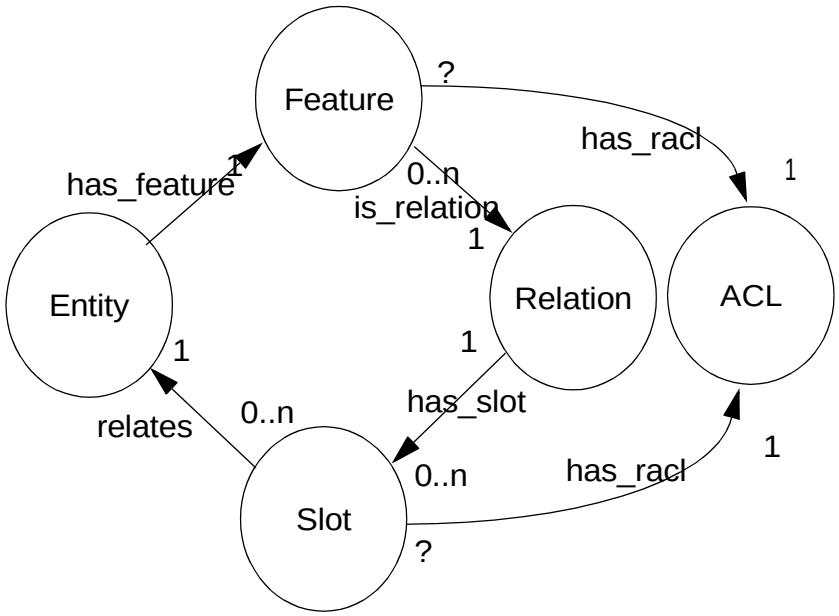

Abbildung 21: Meta-Schema: erweitert um Rollen-ACLs (rACLs)

Die Abbildungen 20 und 21 zeigen diesen Unterschied, die Bedeutung einer solchen Festlegung für konkrete Daten, deren Typschema mit solchen Rollenzugriffsrechten annotiert ist, ist in Abbildung 22 erläutert:

Neben den schwarz dargestellten “normalen” ACLs sind die heller dargestellten rACLs des Datenbankschemas zu erkennen. Diese ACLs beschränken sich nicht nur auf die Ressourcen selbst, sie können auch auf die Relationen zwischen Ressourcen oder Elementen angewandt werden. Als Beispiel aus der Praxis sind in der Abbildung Rollenrechte der Rolle “Autor” eingezeichnet. Der Autor darf natürlich Inhalte bearbeiten und erstellen, nicht aber die Rechtevergabe in den ACLs seiner Ressourcen. Um diese zu ändern, muss er sich erst in der Rolle als “AutorAdmin” anmelden, dann darf er seine ACLs, aber nicht mehr die Inhalte bearbeiten.

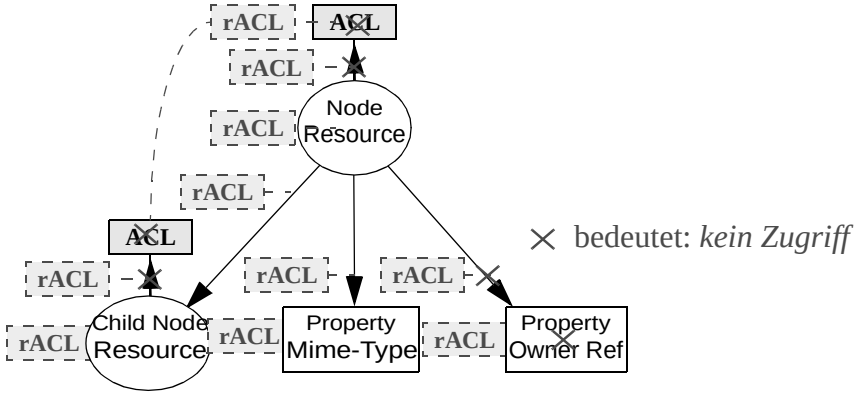

Abbildung 22: Rollen-ACLs für "Autor" und konkrete Inhalte

Für die Benutzer wird das Rollensystem transparent sein. Nach dem Anmelden am System mit Benutzernamen und Paßwort werden alle für den Benutzer eingerichteten Rollen zur Auswahl angegeben. Der Benutzer muss sich daraus temporär eine auswählen. Das Rollenkonzept ist insofern inhärent optional, als jede Abteilung sich eine Rolle ohne jede Einschränkung definieren kann.

Rollendefinitionen auf höheren Organisatonsstufen (einer *Region*, Abschnitt 4.3), sollen Administratoren auf den unteren Ebenen die Arbeit erleichtern und durch die Möglichkeit abteilungsübergreifender Vorgaben andern Benutzern, Autoren, einheitliche Rollen für die gemeinsame Pflege und Benutzung von Inhalten anbieten.

Rollen-ACLs, die im Datenbankschema benutzt werden, haben nur eine verbietende *deny*-Sektion, sie werden nur zur Einschränkung benutzt (Im Datenbankschema kann auf der im Allgemeinen nicht hierarchischen Struktur keine Vererbung der Rechte definiert werden, weswegen *grant*-Einträge die Ausdrucksfähigkeit der Notation nicht erweitern würden).

Bei dem Versuch des Zugriffs auf eine Ressource müssen natürlich die Datenbankschemarechte, rACLs, mit den normalen ACL-Rechten (konjunktiv) kombiniert werden. Erst dann ergeben sich die effektiven Rechte des Benutzers in einer bestimmten Rolle auf die Ressource.

Eine Rollendefinition enthält folgende Elemente:

- Name der Rolle
- Besitzer (Erstellergruppe [Admins])

System-Eigenschaften (Ersteller, Erstelldatum, etc., nur vom System änderbar)

- Eigenschaften (Beschreibung, etc., vom Besitzer änderbar)
- Schema mit Annotationen (rACL-Tripeln)

6.3.2 Systemvorgaben

Anzahl der Rollen, äquivalente Rollen

Da eine Verschiedenheit semantisch gleicher Rollendefinitionen eher weniger zur Koordinierung der Autoren innerhalb eines Departments beiträgt als die Benutzung nur einer der beiden äquivalenten Rollendefinitionen (aber zur Entropie in der Autorengruppe beiträgt), wird festgelegt, dass eine Rolle nur verschieden von vorhandenen Rollen festgelegt werden kann.

Die Neubenennung einer vorhandenen Rolle ist kontraproduktiv im Sinne der Koordinierung der Autoren des Systems, solange sie nicht im Rahmen einer reinen Sprachanpassung geschieht (und damit mit konstantem Aufwand etwa durch Austausch der Sprachressourcen der Autorenprogramme). Sie ist kontraproduktiv, da vorhandene Erfahrung mit der Benutzung einer bekannten, aber umbenannten Rolle eventuell nicht genutzt würde.

Vorgegebene Rollen

Auf der obersten Ebene des vorgeschlagenen Systems werden Rollen vorgegeben. Systemweit notwendige Rollen sind:

- *SystemAdmin*
- *Manager*
- *Author*
- *Consumer* (in diesem Kontext: Student)

Der SystemAdmin ist vom Datenbankschema her nur befugt, die Relationen zwischen Root und Rolle, User, Region und Department zu verfolgen. Dadurch wird verhindert, dass der SystemAdmin Zugriff auf Regions, Departments und/oder Ressourcen erhält. Er ist berechtigt, Benutzer anzulegen und diesen Systemrollen zuzuweisen. Ferner kann er Regions und Departments erstellen und deren Administratoren oder Manager (nur einer pro Department) einsetzen, womit er gleichzeitig den Besitz an dieselben abtritt.

Auf der untersten Ebene wird es vornehmlich die Rolle des Managers und der Department-Admins geben. Der Manager ist vom Datenbankschema her befugt, die Relationen zwischen Department und DepartmentAdmin zu verfolgen. Er ist berechtigt, die Eigenschaften des Department zu ändern und er kann die lokale DepartmentAdmin-Rolle zuweisen. Er kann sich auch selbst zum DepartmentAdmin ernennen. Diese Rollendefinition wird beim Erstellen eines neuen Department automatisch instanziiert und muss benutzt werden.

Der DepartmentAdmin ist befugt, die Relation zwischen Root und Principal, zwischen Department und Group, Department und Rolle sowie zwischen Principal und Rolle und Principal und Group zu verfolgen. Kurz gesagt, wird er in der Lage sein, Benutzer, Gruppen und Rollen zu erstellen und zu verwalten. Er ist außerdem Besitzer der drei Standardäste, darf aber nur die Relationen zwischen Ressourcen und ihren ACLs verfolgen. Der DepartmentAdmin kann die den Besitzer von Ressourcen seines Verwaltungsbereiches verändern. Dies ist dann aber mit einer Systemnachricht an die lokale DepartmentAdminGroup, den Manager und den bisherigen Besitzer verbunden. Dadurch wird mehr Flexibilität erreicht, wenn sich die personelle Struktur unvorhersehbar ändert. Dieses Mittel sollte allerdings die Ausnahme darstellen. Diese Rollendefinition wird beim Erstellen eines neuen Department automatisch instanziiert und muss benutzt werden.

Weitere sinnvolle Rollendefinitionen werden nicht als Systemvorgabe betrachtet, sondern sollten von Administratoren auf höheren Ebenen zur weiteren Instanziierung eingerichtet werden. Solche, zum derzeitigen Zeitpunkt als sinnvoll erachtete Rollen sind:

- ResourceAuthor (entspricht etwa einem Inhaltsautoren aus Abschnitt 6.1.2)
- LayoutAuthor (entspricht einem Designer in HyNoDe, s. Abschnitt 6.1.2)
- PresentationAuthor (entspricht etwa einem Redakteur in HyNoDe)

Der ResourceAuthor ist vom Datenbankschema her befugt auf den Ast Resources zuzugreifen, sowie die Relationen zwischen Ressourcen zu verfolgen. Es ist ihm allerdings nicht erlaubt, die Relationen zwischen Ressourcen und ihren ACLs zu verfolgen. Das bedeutet, dass er zwar die Ressourcen, ihre Kinder und Eigenschaften bearbeiten kann, aber nicht die Zugriffsrechte auf die Ressourcen. Der ResourceAuthor hingegen ist befugt, den Ast Ressource sowie die Relationen zwischen Ressourcen und ihren ACLs zu verfolgen. Das bedeutet, dassmuss er zwar die Zugriffsrechte ändern darf, jedoch nicht die Ressourcen selbst.

Der LayoutAuthor darf den Ast Layouts und die Relationen zwischen Ressourcen und ihren ACLs verfolgen.

Der PresentationAuthor kann sowohl den Ast Resources, Layouts als auch den Ast Presentations verfolgen, sowie die Relationen zwischen Ressourcen. Der PresentationAuthor darf den Ast Presentations und die Relationen zwischen Ressourcen und ihren ACLs verfolgen.

6.3.3 Semantik

Rollen

Rollen sind als Rollen-ACLs über dem Typschema des Datenbestands definiert.

Das Typschema ist ein gerichteter Graph (T_N, T_E), dessen Knoten $T_N = \{\text{typ}(n) | n \in N\}$ die Typen der Knoten aus N sind, die Kanten $T_E = \{\text{typ}(e) | e \in E\}$ die Typen der Kanten aus E.

typ: $N \cup E \to T$ (mit $T_N \cap T_E = \varnothing$)

typ bildet dabei die Hierarchie der Daten auf dieses Schema ab. Der Typ einer Kante $\text{typ}((e_1, e_2))$ verbindet jeweils $\text{typ}(e_1)$ mit $\text{typ}(e_2)$.

Rollen und Rollen-ACLs werden analog zu Principals und ACLs mit Bezug auf bestimmte Datenknoten als $\text{racl}_t \in (R, \{\text{deny}\}, A)^*$ und $(r, \text{deny}, a) \in \text{racl}_t \Rightarrow r \in r(t)$ festgelegt.

Im Unterschied zu den Zugriffsrechten kann auf dem Typschema, das keinen Baum darstellt, keine Menge von ererbten Rollen-ACLs $\text{racl}(t)$ sinnvoll definiert werden. Dieses Fehlen von implizit an einem Typen vorhandenen Rollen-ACLs macht die Verwendung von *grant* und *deny* sinnlos. Deswegen wird vereinfacht bei Rollen-ACLs nur *deny* verwendet und *grant* durch das Fehlen einer entsprechended deny-Rollen-ACL formuliert.

Rollenzugriff

Der Zugriff a eines Principals p in Rolle r auf einen Knoten n der Datenhierarchie ist dann erlaubt, wenn die Aktion a auf diesem Knoten nicht für Rolle r verboten ist. Ferner darf diese Aktion auch auf keinem seiner Ahnen bis inklusive zum ersten Ahnen-Knoten, der explizit für Principal p (oder eine Gruppe, deren Mitglied p ist) verboten sein. Diese innerhalb eines Rollensystems erlaubten Zugriffe sind durch die Abbildung $\text{raccess}: N \to P \times R \times A$ gegeben, die in der folgenden Festlegung (2) defniniert ist.

$$
\begin{aligned}
\text{raccess}(n) = \{(p, r, a) \mid\ & \\
& (p, a) \in \text{access}(n) \\
& \wedge ((r, \text{deny}, a) \notin \text{racl}(\text{type}(n))) \\
& \wedge (((p, \text{grant}, a) \in \text{acl}(n)) \\
& \quad \vee ((\exists n', (n, n') \in E \wedge (p, r, a) \in \text{raccess}(n'))) \\
& \quad \wedge ((r, \text{deny}, a) \notin \text{racl}(\text{type}((n', n))))))) \\
\}\ &
\end{aligned}
\tag{2}
$$

Für Referenzen (auf andere Knoten), bedeutet dies, dass sie eventuell geändert werden können, ohne dass der Zielknoten geändert werden kann, denn Referenzen selbst werden auch als editierbare und verwaltbare Attribute behandelt.

6.4 Bewertung des Rollenkonzeptes

Das vorgeschlagene Rechtekonzept bietet die Möglichkeit, die notwendigen Rollen der beiden Applikationen aus Kapitel 6.1.1 und Kapitel 6.1.2 zu realisieren.

Daneben erlaubt es generisch die dynamische Realisierung einer ganzen Klasse von Rollen, die beim dezentralen Erstellen, Verwalten und Wiederbenutzen multimedialer Inhalte auftreten können. Diese Klasse sind Rollen, die allgemein im Rahmen des Pflegens von solchen Inhalten im Umfeld einer Organisationsstruktur wie etwa von Lehrstühlen auftreten.

Die Einschränkung hier ist im Bereich weitergehender Semantik zu finden, insbesondere der Semantik der erzeugten Inhalte des Systems. Soll diese mitverwendet werden, sollen etwa in einem Vorlesungsarchiv speziell für Mediziner die Rollen des Dentisten, des Anästhesisten und des HNO-Arztes realisiert werden, so reicht das oben angegebene Konzept nicht aus.

Eine weitere Möglichkeit neben der Spezialisierung des Datentyps auf ein bestimmtes inhaltliches Thema ist eine Erweiterung der Rollenbeschreibung. Die Rollenbeschreibung verknüpft Typ und Inhalt der Daten dadurch, dass sie Beschreibungen des Datentyps der Anwendung in deren Inhalt ablegt (und annotiert). Beides muss vom System konsistent gehalten werden. Die Beschreibung des annotierten Datentyps aber könnte durch Beschreibungen des möglichen Inhaltes der Instanzen des jeweiligen Datentyps erweitert bzw. spezialisiert werden, hier ist eine ganze Skala von einfachen, formalen Ausdrücken bis hin zu integrierten externen Programmteilen oder Skripten denkbar. Diese Erweiterung des Rollenkonzeptes könnte auch benutzt werden, gänzlich andere Rollenkonzepte zu realisieren.

Der hier vorgestellte eingeschränkte Ansatz wurde vor allem nach drei Kriterien ausgewählt:

- Einfachheit
  Das Konzept überdeckt die notwendige Funktionalität der drei vorgestellten Echt-Welt-Applikationen, ohne Darüberhinausführendes einzuführen. Nur die gewählte Abstraktion wurde durch die lokale, dezentrale Administration der Arbeitsvorgänge des dritten Beispiels notwendig.
- Generizität
  Der Ansatz ist für Medien aus allen Zielbereichen anwendbar, er ist bezüglich der Medieninhalte thematisch nicht festgelegt.

Weiterhin ist er durch eine klar definierte Erweiterung (Bewertung der Daten-Typ-Instanzen durch weitere Annotierung der Schemata der Rollendefinitionen) qualitativ erweiterbar wenn notwendig.

- Deutlichkeit, Transparenz und Orthogonalität
  Das hier vorgestellte Rollenkonzept stellt eine deklarative Definition der Rollen zur Verfügung, im Gegensatz zu einer prozedural kodierten wie im Beispiel in Abschnitt 6.1.1 (oder eines durch prozedurale Algorithmen annotierten Schemas).
  Weiterhin wirkt sich das vorgestellte Konzept systemweit auf alle Inhalte gleich aus und ist dadurch weniger schwierig zu verwenden. Den gleichen Effekt hat die garantierte Orthogonalität der Rollendefinitionen zu den Rechtefestlegungen: eine Erweiterung, die mehr als die Typ-Information benutzt, würde hier schwer nachvollziehbare Inferenzen zwischen Rechte- und Rollenfestlegungen erlauben (vrgl. Abschnitt 6.4.2).

6.4.1 Machbarkeit

Eine Implementierung der Semantik aus Abschnitt 6.3.3 in SWI-Prolog über einem gegenüber Kapitel 4.3 vereinfachten Datenbestand erfüllte die in Kapitel 6.2 gefundenen Anforderungen. Eine solche Implementierung realisiert zusammen mit den erforderlichen Annotationen mit Rollen-Informationen (Kapitel 6.3.3) obige Spezifikation.

6.4.2 Vollständigkeit

Für jeden Knoten, auf den ein Benutzer nach Auswertung der normalen Zugriffsrechte hat, muss ein *grant*-ACL-Eintrag für diesen Knoten oder einen seiner Ahnen existieren. Aus Festlegung folgt damit unmittelbar, dass für diese Knoten die Erlaubnis des Rollen-Zugriffs *raccess* in endlicher Zeit überprüft werden kann, indem bis zum explizit zugänglichen Ahnenknoten die Abwesenheit eines *deny*-rACL-Eintrages überprüft wird.

6.4.3 Komplexität

Für den Aufwand der Berechnung der Rollenerlaubnis auf ein Datenelement soll die Anzahl der dazu betrachteten Datenelemente gezählt werden:

Für einen Zugriff (p, r, a) auf ein beliebiges, initiales Datenelement n müssen zusätzlich zum Aufwand für die Auswertung der (r)ACL-Tupel von n eventuell die aller Väter ausgewertet werden. Der entsprechende Aufwand ist proportional zur Baumtiefe, also logarithmisch zur Größe des Gesamtsystems. Die Komplexität des Zugriffs erhöht sich also nicht prinzipiell durch ein Rollenkonzept, das nur die Datentypinformation benutzt. Allein die Identifizierung eines Datenelements hat bei effizienter Suche diese Komplexität.

Durch die (durchaus sinnvolle) Festlegung auf eine bestimmte, kleine Anzahl von möglichen Rollen für jede Verwaltungseinheit kann der konkrete Aufwand noch drastisch eingeschränkt werden, wie die folgende Überlegung zeigt.

Ein Arbeitshandbuch eines Lehrstuhls, das sehr viele Rollen beschreibt, müsste sehr detaillierte Vorgaben machen und sehr feine Unterscheidungen plausibel machen, jedoch ohne anhand konkreter Inhaltsdaten Rollen differenzieren zu können. Eine Anzahl von 50 Rollen ist

hier ganz klar in der Praxis unsinnig, eine Obergrenze dieser Anzahl von etwa 10 pro Abteilung aber gut zu vertreten. Die recht komplexen Applikationsbeispiele in Abschnitt 6.1.1 und Abschnitt 6.1.2 benötigten jeweils vier bzw. fünf aufeinander abgestimmte Rollen, die Rolle der reinen Konsumenten jeweils mitgerechnet.

Kapitel 7 - Versionierung

In diesem Kapitel wird ein weiterer Aspekt bearbeitet, der beim Erstellen und Bearbeiten von Inhalten relevant ist: die Versionierung von Inhalten.

Ein einfaches Beispiel einer Versionierungsfunktion, die Autoren beim Bearbeiten von Ressourcen zur Verfügung haben können, ist die "Undo"-Funktion, das Ersetzen des Ressourcenzustands mit dem Zustand vor dem letzten Bearbeitungsschritt. Diese und viel auch weitergehende Funktionen zur Verwaltung verschiedener Versionen finden sich als nützliche und notwendige Systemfähigkeiten dort, wo die jeweils bearbeiteten Ressourcen ein gewünschtes Arbeitsergebnis darstellen, das durch die Verwendung eines Programms generiert werden soll.

Ziel dieses Kapitels ist es, zu untersuchen, welche Versionierungsfunktionalitäten in welcher Form hier, also im Umfeld verteilter, multimedialer Applikationen mit vielen Autoren, gewünscht und sinnvoll sind. Dazu werden zunächst in Abschnitt 7.1 Szenarien präsentiert, wie sie so von Autoren für sinnvoll gehalten werden. Dann werden in Abschnitt 7.2 die in Abschnitt 3.3 vorgestellten Systeme auf ihre Eignung untersucht, diese Szenarien zu unterstützen. Insbesondere wird untersucht, ob und welche bestehenden Mechanismen übernommen werden können. In Abschnitt 7.3 wird ein Zielsystem entwickelt, das die Anforderungen der Szenarien unterstützt, und vorhandene Mechanismen werden entsprechend angepasst. Im Abschnitt 7.4 wird ein solches System prototypisch entwickelt.

7.1 Zielanalyse

Überall, wo Ressourcen über die Zeit bearbeitet und wiederbenutzt werden, wächst der Bedarf an maschineller Unterstützung der Pflege und Verwaltung dieser Ressourcen, also auch für Versionierungsmechanismen, mit dem Wert der Ressourcen. Dieser Wert bestimmt sich auch durch den Aufwand, den Ressourcenpflege und Verwaltung kosten.

Wenn Ressourcen durch viele Nutzer autonom bearbeitet werden und gemeinsam genutzt werden können, wächst der Aufwand mit der Zahl der Nutzer, mit der Aufwändigkeit der Ressourcen (etwa bei der Erstellung von Videos) und der Komplexität der Abhängigkeiten zwischen den Ressourcen (wie etwa bei Hypertext-Dokumenten).

Die in Abschnitt 3.3 vorgestellten Ziele von Versionierung werden ich hier noch einmal aus Nutzersicht zusammengefasst:

Historie der Inhalte: Dem Nutzer wird ermöglicht, unter mehreren Versionen individuell wählen zu können, d.h. eine neue Präsentation ersetzt eine alte Präsentation nicht unwiderruflich, sondern ergänzt den Bestand an Präsentationsversionen.
Dies ist das eigentliche und hauptsächliche Ziel von Versionierung.

Konfliktvermeidung: Die Aktualisierung von Inhalten wird durch die Nutzer unabhängig voneinander vorgenommen. Es sind Mechanismen notwendig, die ein Durcheinander der Aktualisierungen verhindern. Nutzer müssen von den Änderungen erfahren können.
Diese Aufgaben ergeben sich, wenn mehrere Arbeitsschritte gleichzeitig, etwa durch verschiedene Nutzer, an voneinander abhängigen Inhalten vorgenommen werden.

Individueller Arbeitskontext: In Systemen, die Nutzern einen individuellen Arbeitskontext zu Verfügung stellen, ist auch dieser Bereich von der Existenz der Versionierung betroffen, wo diese vorhanden ist. Hier muss der Kontext neben Verweisen auf bearbeitete Inhalte nun auch Versionsinformation enthalten. Etwa, damit jeder Nutzer seine Aktionen individuell zurückverfolgen kann. Falls der Wunsch besteht, vorher benutzte Zustände der Inhalte wiederzuerlangen, sollten diese einfach zu erreichen sein.

Im folgenden werden aus einer Betrachtung der wichtigsten Nutzungsszenarien die speziellen Anforderungen an ein Versionierungssystem für ein wie in Abschnitt 2.1.3 beschriebenes verteiltes Vorlesungsarchiv erarbeitet. Das darin addressierte Problemfeld stammt aus dem direkten Erfahrungsbereich der befragten Autoren, um möglichst realistische Szenarien zu erhalten.

Weiter stellt dieses Beispiel ein komplexeres Umfeld als die beiden anderen Beispiele aus Abschnitt 2.1 dar, also eine Versionierungskonzept hierfür weitgehend durch Vereinfachung an die beiden anderen Applikationen adaptiert werden kann.

7.1.1 Nutzungsszenarien

In diesem Abschnitt sollen Umfang und Form von Versionierungsfunktionalitäten gefunden werden, wie sie von Nutzern gewünscht und akzeptiert werden. Dies soll anhand von Szenarien geschehen.

Die folgenden Szenarien wurden wieder aus den typischen Anwendungssituationen abgeleitet, wie sie beim Erstellen gemeinsam genutzter multimedialer Lehrinhalte in verschiedenen Teams auftreten. Die Szenarien wurden in Diskussionen mit den Teams der in Kapitel 2 erwähnten Projekte [51][9][34][47] als die relevanten und kennzeichnenden Szenarien identifiziert. Die folgende Liste beginnt beim einfachsten aber wichtigen Szenario des individuellen Editierens und berücksichtigt schrittweise weitere Aspekte, die den Einsatz und möglichen Nutzen von Versionierung bei Fehlern und Konflikte (etwa durch die Verteilung der Inhalte) berücksichtigen.

Es ist natürlich nicht möglich, hier eine komplette Liste von sicher zutreffenden Szenarien mit Benutzerverhalten a priori zu konstruieren. Vielmehr wird mit diesen Szenarien beabsichtigt, durch die schrittweise Verfeinerung eine realistische Sicht auf die Anforderungen durch die Nutzer zu bekommen, die einen weiten Teil der möglichen Nutzung abdeckt und die wichtigsten Aspekte erfasst.

Szenario 1: Benutzen der aktuellen Ressourcen. Prof. Maier bearbeitet seine Vorlesung “Ausgewählte Kapitel der Technologie Integrierter Schaltungen”. Da er diese Vorlesung weiterentwickelt und ein Arbeitsschritt jeweils auf dem anderen aufbaut, will er nur die aktuelle Version sehen und verändern.
Beim Abruf der Präsentation kann er sicher sein, dass diese immer die neuesten Versio-

nen ihrer Inhalte zeigt.
Versionierungsinformationen und -vorgänge sollen hier, während des Normalbetriebs, höchstens für die Arbeit des Autoren nicht notwendigerweise und nur wenn gewünscht sichtbar sein.

Wegen der normalerweise kleinen Teams der Zielapplikation, die gleichzeitig an einer Vorlesung arbeiten, und wegen der Notwendigkeit des konzentrierten Arbeitens, stellt dieses Szenario den Normalfall der Nutzung dar.

Die Aktualisierung soll in diesem Szenario automatisch laufen. Der Server sucht sich jeweils immer die aktuellen Ressourcen zur Erstellung der Präsentation heraus. In diesem Zusammenhang ist genau festzulegen, welche Version einer Ressource immer die aktuelle ist. Versionsbäume von Ressourcen und andere eventuell anfallenden Versionsinformationen müssen vor dem Nutzer verborgen werden können, ohne eine verdeckte Versionierung (für die restlichen Szenarien) unmöglich zu machen.

Dies erfordert insbesondere eine gewisse Unabhängigkeit von Ressourcen-Identifikation und Versionsidentifikation. So muss sich der ganze Inhaltsbestand auf eine Basis von Tupeln aus Ressourcen-ID und Versionsbezeichnern abbilden lassen, und zum andern muss das System zu einer gegebenen Ressource den ganzen Graphen der Versionen und Varianten selbständig bestimmen können.

Die Bestimmung einer aktuellen Version ist trivial, wo es nur jeweils vollständig geordnete Versionen einer Ressource wie in diesem Szenario gibt. In den weiteren Szenarien wird dies zu erweitern sein, wenn die vollständige Ordnung über Instanzen einer Ressource durch Einführung von konkurrierenden Varianten verloren geht.

Weitere Aspekte, die dieses eigentlich triviale erste Szenario von einem Szenario ohne Versionierung unterscheiden, sind in der eigentlichen Arbeit der Autoren nur implizit, aber dennoch wichtig. So muss er zum einen wissen, dass gelöschte Inhalte (hier: Versionen) im System erhalten bleiben, was ihm das System aus datenschutzrechtlichen Gründen auf anderen Kanälen explizit mitteilen muss. Ebenso ist es sinnvoll, dem Autoren auch aus Gründen der Schonung von Systemspeicher- und allgemeiner Kapazität Kenntnis darüber zu vermitteln, dass seine Arbeit nicht mehr nur verändert, sondern immer den Systeminhalt vergrößert (Ein eventuelles *Pruning*, also Löschen oder Auslagern von Inhalten, berührt die möglichst einfach gehaltene Interaktion des Autoren mit dem System in diesem Szenario nicht. Pruning sollte am besten transparent vom System oder den Systemverwaltern vorgenommen werden).

Szenario 2: Benutzen einer alten Version einer Ressource. Prof. Maier benutzt in seiner Vorlesung ein Schaubild, um den Stand der Technik im Jahr der Erstellung der Vorlesung zu demonstrieren. Spätere Beispiele sollen eventuelle Fortschritte aufzeigen.
Hierzu legt er im System fest, dass auch Aktualisierungen an diesem Schaubild keine Auswirkung auf seine Präsentation haben dürfen, dass immer die alte Version verwendet wird.

Versionierungsinformationen sind hier nur insofern für den Nutzer wichtig, als er einer (Verwaltungs- oder Kontroll-) Darstellung der Präsentation entnehmen können muss, dass die verwendete Ressource eine *feste*, und keine dynamische angepasste Version ist. Zur Identifikation einer solchen verwendeten festen Version muss der Nutzer auf den Versionen navigieren und

suchen können, zum Beispiel über einer Repräsentation der Versionzusammenhänge als Graph.

Das Verwenden von (aktuellen oder alten) Inhalten kommt auch zwischen Ressourcen und Ressourcen und Layouts vor.

Szenario 3: Zurücksetzen auf eine alte Version einer Ressource. Das Szenario 1 ergänzend ist der wichtige Ausnahmefall, dass ein oder mehrere Arbeitsschritte zurückgenommen werden.

Das System kann hier zwei Alternativen anbieten: das unwiderrufliche Löschen des Ergebnisses einer Sequenz von normalerweise unmittelbar vorher erfolgten Arbeitsschritten oder das Nutzen eines Ressourcenzustandes vor Arbeitsschritten, die auch beliebig weit zurück liegen können.

Zunächst werden die beiden Alternativen in Szenario 2 (a) und Szenario 2 (b) genauer beschrieben und dann gemeinsam kurz diskutiert.

Szenario 3(a): Zurücknehmen von Arbeitsschritten: *Undo*. Unzufrieden mit dem aktuellen Stand des soeben im System geänderten Lehrvideos, nimmt Professor Maier den letzten Bearbeitungsschritt daran zurück. Im System steht nunmehr nur noch die vormalig aktuelle Version des Videos zur Verfügung. Diesen *undo*-Schritt könnte er auch wiederholen, um eine ganze Sequenz von Schritten zu tilgen, wobei das System zum einen dabei verhindert, dass er eine benutzte Version löscht, und zum andern Prof. Maier die Tragweite seiner Entscheidungen klar mitteilt.
Will er einen Schritt nicht komplett verwerfen, kann er entsprechend Szenario 3(b vorgehen.

Szenario 3(b): Alternative Sequenzen von Arbeitsschritten. Hier soll eine eine alternative Variante zusätzlich zur aktuellen Version einer Ressource erzeugt werden.
Professor Maier identifiziert die ältere Version, *v1*, eines Lehrvideos und "editiert" diese, d.h., er erzeugt eine Variante des Videos, die als Alternative zu den bisherigen Nachfolgeversionen nun von *v1* vom System zur Verfügung vorgehalten wird. Zu beachten ist hier, dass es nun zwei "aktuelle" Versionen des Lehrvideos im System gibt.
Offensichtlich führt diese Einführung von Varianten im Allgemeinen zu einem Baum von Versionen und Varianten.

Da hier von einem Nutzen der Versionierung gebrauch gemacht wirdDa der Nutzer die automatische Aktualisierung nicht mehr wünscht, muss diese für ihn auch ein- und abschaltbar sein. Dies ist die erste Anforderung, die sich aus diesem Szenario ableiten lässt. Es muss auch ermöglicht werden, einen bestimmten Inhalt der Vorlesung für die Zukunft festzulegen. Damit der Nutzer dies auch machen und selektieren kann, müssen Funktionen vorhanden sein, die es dem Professor ermöglichen, eine gezielte Selektion der Ressourcen nach intuitiven Kriterien durchzuführen. Bezüglich der benötigten Funktionalitäten lassen sich somit folgende Punkte aufzählen:

1. *Identifikation einer Version der Ressource:*
   Durch geeignete Datenspeicherung sollte gewährleistet sein, dass es möglich ist, durch eine Suchfunktion eine Auswahl zwischen den einzelnen Varianten derselben Ressource zu treffen und diese Auswahl zu fixieren. Hierzu werden die Versionen von Ressourcen in einem Versionenbaum gespeichert, in dem die Versionsnummern sowie Vorgänger- und Nachfol-

gerbeziehungen zwischen den Versionen einer Ressource enthalten sind. Damit der Nutzer auch nach diesen Versionen einer Ressource suchen kann, ist eine graphische Benutzeroberfläche nötig, die eine Auswahl der Versionen einer Ressource auf intuitive Art ermöglicht.

2. *Aktualisierung:*
   Der Nutzer kann zwischen einer automatischen Aktualisierung, welche die Präsentation mit den aktuellen Ressourcen erstellt, und einer manuellen Aktualisierung auswählen, die der Nutzer nur bei Bedarf durchführen kann. Dies impliziert für die Datenstruktur, dass die Referenzen auf Ressourcen dynamisch gemacht werden müssen. Bei einer statischen Referenz wird immer derselbe konstante Zustand einer Ressource verwendet, unabhängig von weiterhin erfolgenden Änderungen. Dynamische Referenzen jedoch werden diese immer auf dem aktuellen Stand halten, was bedeutet, dass mit jeder Änderung einer Ressource sich der neue Zustand der Ressource sich in der Darstellung der verweisenden Resource spiegelt.
   Ähnlich verhält es sich mit dem Arbeitskontext jeden Benutzers. Auch hier muss entschieden werden können, ob eine feste Version einer Ressource, oder der jeweils neueste Zustand einer Ressource Gegenstadt der Arbeiten des Nutzers ist. DasVersionierungs-System muss also persistent festhalten, welche Präsentationen der Nutzer benutzt hat und welche Änderungen vorgenommen wurden. Dies könnte mit einem individuellen Nutzer-Profil geschehen.
3. *Aktualisierungs- und Präsentationsmodus:*
   Damit Nutzer Ihre gewünschten Änderungen in den Präsentationen durchführen können, sollte eine Art Kontroll- oder Editiermodus neben dem Präsentationsmodus eingeführt werden, in dem der Nutzer seine Änderungen an der Präsentation oder an seinen eigenen Einstellungen vornehmen kann.

Szenario 4: Entferntes Update einer Ressource. Prof. Mayer benutzt auch fremde Ressourcen, etwa von Prof. Mayer. Prof. Mayer aktualisiert ein Schaubild, das Prof. Müller benutzt. Da Müller immer die aktuellen Inhalte für seine Präsentation benutzt, wird diese nun für Ihn bereitgestellt. Prof. Mayer will, da dieses eine Änderung seiner Vorlesung darstellt, aber informiert werden und entscheiden, ob er diese Änderung annimmt, bevor er in den Vorlesungssaal geht.Dazu bekommt er eine E-Mail mit einer Zusammenfassung aller für ihn relevanten Änderungen durch Prof. Mayer nach dessen Änderungssitzung, außerdem kann er in der Kontrollsicht auf die Präsentation anliegende Änderungen erkennen, annehmen oder auch verwerfen (und damit die alte Version des Schaubilds für seine Vorlesung festlegen).

Diese Situation ist typisch für Mehrbenutzersysteme. Ein Nutzer nimmt unabhängig von Anderen Änderungen an der Datenbank vor. Dies kann dazu führen, dass Nutzer der aktuellen Version (dynamische Referenzen) dieser Ressource mit der Änderung überrascht werden. Damit dies vermieden wird, müssen diese Anwender aktiv über Updates informiert werden. Diese Form der Benachrichtigung wird auch Push-Benachrichtigung genannt.

Für Nutzer von Präsentationen, welche auf bestimmte Versionen von Ressourcen mit statischen Referenzen verweisen, ist die aktive Benachrichtigung nicht relevant, da diese bei Änderungen nicht betroffen sind. Damit diese trotzdem über Aktualisierungen informiert werden, sollte es noch eine zweite Art der Benachrichtigung geben, die bei Bedarf über neuere Versionen in Kenntnis setzt. Dies wäre damit die passive Art der Update-Benachrichtigung, die nur bei Bedarfsanmeldung durch den Nutzer geschieht (Pull-Benachrichtigung).

Die Information über Änderungen geschieht am einfachsten und effektivsten per elektronischer Post, da diese einfach zu generieren, für jedermann zugänglich und einfach zu verteilen ist. Da auf der anderen Seite bei vielen Updates die Anzahl der erhaltenen E-mails für den einzelnen Anwender lästig werden kann, muss noch eine zweite Form der Benachrichtigung vorhanden sein. Dabei ist auf detaillierte Informationen zu verzichten. Stattdessen sollte für jede veränderte Ressource eine einfache Statusanzeige vorgenommen werden, die nur anzeigt, dass eine Änderung vorgenommen wurde. Dies geschieht am besten direkt beim Einloggen des Nutzers in das System durch Anzeige einer Warnlampe und im Kontrollmodus (siehe Szenario 4) durch Anzeigen von Warnlampen für jede Ressource, die geändert wurde. Der Nutzer kann die Präsentation dann immer noch begutachten und die Änderungen übernehmen oder verwerfen, indem er auf eine ältere Ressource zurücksetzt.

Zusammenfassend ist dieses einfache Modell der Benachrichtigung bei entfernten Updates in Tabelle 5 dargestellt.

Tabelle 5: Benachrichtigungen bei entfernten Updates

Form der Benachrichtigung	Benachrichtigungsmodus	Referenzart			
		dynamisch		statisch	
		voreingestellt	umschaltbar	voreingestellt	umschaltbar
E-Mail nach Update	Push	X	X		X
Globale Warnlampe vor Start der Präsentation	Pull	X			X
Warnlampen im Kontrollmodus	Pull	X			X

Szenario 5: Festlegen einer bestimmten Präsentation. Prof. Mayer will seine Vorlesung in der jetzigen Version festschreiben, da er die Vorlesung an zwei Hochschulen hält und eine einheitliche, bekannte Version dieser Vorlesung für die Prüfungen später wichtig ist. Dazu werden die Versionen aller verwendeten Ressourcen in dieser Präsentation festgeschrieben, und die Version der Präsentation mit den festen Ressourcen wird im Nutzerkontext von Prof. Mayer notiert.

Nutzer-Profile verweisen auf die entsprechenden Präsentationen.

Wenn ein Nutzer eine bestimmte Präsentation für die Zukunft fixiert, so müssen die Referenzen der Präsentation auf die Ressourcen als statisch festgelegt werden, das heißt diese dürfen nicht mehr auf die aktuelle Version einer Ressource verweisen. Somit entsteht eine neue Version der Präsentation, die in der Präsentationsliste abgespeichert wird und dessen Versionsvorgänger die vorher benutzte Präsentation ist. Desweiteren muss im Nutzerprofil auf diese neue Version der Präsentation mit einer statischen Referenz verwiesen werden. So ist gewähr-

leistet, dass immer auf dieselbe Präsentation verwiesen wird, diese wiederum immer auf dieselben Ressourcen verweist.

Will der Nutzer auf eine vorher von ihm benutzte Präsentation zurückgreifen, so muss hierfür im Nutzerprofil eine Liste (History) für bisher benutzte Präsentationen erstellt werden. Bei einer Anfrage seitens des Nutzers zum Zurücksetzen auf eine ältere, von ihm benutzte Präsentation kann anhand der History-Liste bestimmt werden, welche Präsentation benutzt wurde. Beispielsweise ist eine “Undo”-Funktion hiermit denkbar, durch die eine Anweisung an die Datenbank geschickt wird, die dafür sorgen soll, dass die Präsentation, die er vorher benutzt hat, geladen wird. Das Versionierungssytem wird dann im Nutzer Kontext auf den Vorgänger der aktuell benutzten Präsentation umstellen. Dies geht natürlich auch umgekehrt, indem auf den Nachfolger der aktuellen Präsentation umgestellt wird.

Diese Änderungen müssen ebenfalls in dem oben schon geforderten Kontrollmodus laufen, und Nutzer dieser Präsentation müssen benachrichtigt werden.

Szenario 6: Aktualisierung einer festgelegten Präsentation. Prof. Mayer editiert seine vorher festgelegte Präsentation. Für ein Videofile wünscht er sich immer die aktuelle Version. Andere Teile der Präsentation jedoch sollen so bleiben wie bisher.

In der Präsentationsliste verweisen die einzelnen Präsentationselemente auf Ressourcen per Referenz. Hier tritt der Fall auf, dass ein Nutzer die Referenz auf eine Präsentationsressource dynamisiert und alle anderen erhalten bleiben. Dies hat die Folge, dass eine neue Präsentation in der Präsentationsliste eingefügt wird, nachdem der Nutzer diese Präsentation für die allgemeine Nutzung freigibt. Der Vorgänger dieser neuen Präsentation ist die vorher benutzte, mit dem Unterschied, dass hier eine oder mehrere Referenzen dynamisiert wurden. Somit wird hier auch eine Versionsspeicherung der Präsentationen in der Präsentationsliste notwendig.

Die Editierung der Referenzen geschieht ebenfalls im Kontrollmodus. Hier sollte der Nutzer auch die Gelegenheit haben, einen Überblick über die verschiedenen Versionen einer Ressource zu erhalten.

Szenario 7: Auswahl einer bestimmten Präsentation. Prof. Hubert möchte sich probeweise verschiedene Präsentationen anschauen und eine für die zukünftige Benutzung auswählen.

Zu diesem Zweck wechselt der Professor in den Kontrollmodus. Hier sucht er sich eine Präsentation aus der Präsentationsliste aus und fährt einen Probelauf. Da Präsentationen auch in verschiedenen Versionen vorkommen, ist hier auch eine graphische Repräsentation der Versionsstruktur notwendig, in welcher der Professor eine Auswahl treffen kann.

Szenario 8: Aktualisierung des Formats für Präsentation. Prof. Stein ist mit seiner Vorlesung zwar zufrieden, jedoch hat sich das Logo der Universität geändert. Daher will er für seine Vorlesung das aktuelle Format haben. Er verändert das Format, das seine Vorlesungen benutzen und betrachtet die sich nun ergebende aktuelle Präsentation seiner Vorlesung mit dem neuen Logo.

Dieses Szenario hat keine Änderung der Inhalts-Ressourcen zur Folge. Hier ändert sich nur das benutzte Format, d.h. implizit auch die Präsentation, die nun auf eine andere Layout-Ressource verweist. Da der Nutzer hier sein benutztes Format ändert und davon immer die aktuelle Ver-

sion haben möchte, bietet es sich an, dass die Formate für Präsentationen auch in verschiedenen Versionen vorliegen. Gleichzeitig müssen wie bei den Ressourcen die Änderungen von Formaten den Nutzern mitgeteilt werden. Wie bei der Auswahl der Ressourcen auch, so muss auch bei der Auswahl der Formate eine Möglichkeit gegeben sein, immer nur eine bestimmte Version der Formate zu benutzen. Hierzu ist auch für die Formatdefinitionen ein Versionsbaum bereitzustellen.

7.1.2 Fazit: Auswertung der Szenarien

Tabelle 6: Nutzungszenarien und abgeleitete benötigte Funktionalität

Nutzungsszenarien		Erforderliche Funktionalität
1	Benutzen der aktuellen Ressourcen	• Versionsbaum für Ressourcen • Aktualisierungsvorgang bei Änderung an den Ressourcen • Benachrichtigung bei Änderungen
3	Zurücksetzen auf alte Version einer Ressource	• Ein- und Ausschalten der automatischen Aktualisierung • Bereitstellung eines Versionsbaumes zur Identifikation von Versionen der benutzten Ressourcen • Kontroll- bzw. Editiermodus zur Bearbeitung der Präsentation oder der persönlichen Einstellungen • Nutzerprofil zur Speicherung der bisher benutzten Präsentationen
4	Entferntes Update einer Ressource	• Push-Benachrichtigung aller Nutzer der Präsentation, welches dynamisch auf die aktuelle Version einer Ressource referenziert • Pull-Benachrichtigung aller Nutzer der Präsentation, welche statisch auf eine bestimmte Version einer Ressource verweist • Form: E-mail oder Indikation der Änderungen
5	Festlegen einer bestimmten Präsentation	• Versionsbaum für Präsentationen • Konfigurierbare Referenzen auf Ressourcen und Präsentationen • Dynamische Referenzen: Zeigen immer auf aktuelle Version; werden vom System aktualisiert • Statische Referenzen: Zeigen nur auf eine bestimmte Version • Nutzer-Kontext zur Rückverfolgung der bisher benutzten Präsentationen
6	Aktualisierung einer festgelegten Präsentation	• Konfigurierung der Referenzen • Editierung im Kontrollmodus • Aktualisierung des Versionsbaumes für Präsentationen
7	Auswahl einer bestimmten Präsentation	• Versionsbaum für Präsentationen • Editiermodus
8	Aktualisierung des Formats für Präsentationen	• Versionsbaum für Layouts • Benachrichtigung bei Updates

Im folgenden Abschnitt werden die vorgestellten Szenarien in Einzelfunktionalitäten dekomponiert und die identifizierten benötigten Funktionalitäten gesammelt. Im weiteren Kapitel 7 wird dann die Verwirklichung dieser Funktionalitäten addressiert.

Zur Identifikation der Ziele einer Versionierung für das Authoring eines multimedialen Systems wie in Abschnitt 2.1.3 fasst Tabelle 6 die Ergebnisse der Szenarien aus Abschnitt 7.1.1 zusammen.

Die ermittelten Funktionalitäten sind notwendig, um die beschriebenen Szenarien zu erfüllen. Da für ein Szenario nie eine einzige Funktion benötigt wird, werden in der nächsten Tabelle 7 die benötigten Funktionalitäten einzeln aufgeführt und den Szenarien zugeordnet. Mit der fol-

Tabelle 7: Benötigte Funktionalität

Funktionalität		Anwendung in Szenario Nr. (Tabelle 6):						
		1	**3**	**4**	**5**	**7**	**6**	**8**
1	Versionsbaum für ...	**X**						
	... Präsentationen				**X**	**X**	**X**	
	... Ressourcen	**X**	**X**	**X**				
	... Layouts							**X**
2	Nutzernotifikation bei entfernten Updates	**X**						
	Push-Notifikation	**X**		**X**				**X**
	Pull-Notifikation	**X**						
3	Systemseitige Aktualisierung der Präsentationen bei Änderungen der Ressourcen, Layouts oder Präsentationen	**X**	**X**					
4	Ein- und Ausschalten der Aktualisierung		**X**		**X**			
5	Kontrollmodus		**X**	**X**	**X**	**X**	**X**	**X**
6	Konfigurierbare Referenzen	**X**						
	Dynamisch	**X**				**X**		**X**
	Statisch		**X**		**X**	**X**		
7	Nutzer-Kontext		**X**		**X**	**X**	**X**	**X**

genden Tabelle 7 kann auch direkt bestimmt werden, welche Funktionalitäten aus den CM- und DBMS-Systemen (s. Abschnitt 3.3) übernommen werden.

In Tabelle 7 ist der Umstand hervorzuheben, dass für den Normalfall der Nutzung (Szenario 1) die erforderlichen Funktionalitäten sehr begrenzt bleiben. Aufgrund der anderen Szenarien haben sich Anforderungen ergeben, die diesen Normalfall ergänzen, insbesondere durch Einführung eines Kontroll- bzw. Editiermodus. Die wichtigsten Erweiterungen zur Datenstruktur aus Abschnitt 4.3 sind somit die Bereitstellung von Versionsbäumen für die verschiedenen Elemente der Präsentationen, der Kontrollmodus, ein Nutzerprofil, Nutzernotifikation, sowie dynamische Referenzen. Aus den oben diskutierten typischen Anwendungsszenarien lassen sich somit Bereiche der bisherigen Datenstruktur herauskristallisieren, die um zusätzliche Elemente ergänzt werden müssen.

Tabelle 8: Zu erweiternde Bereiche der Datenstruktur

Elemente der Datenstruktur	Notwendige Datenstrukturerweiterungen
Ressourcen/ Präsentationen/ Layouts	• Versionsbaum mit Vorgänger- und Nachfolgerbeziehungen

Referenzen	• Dynamisierung der Referenzen durch Definition eines Types: "current"-Typ für dynamische und "frozen"-Typ für statische Referenzen
	• Dynamisierung in fünf Bereichen:
	  • Ressourcen-Referenzen in Präsentationen
	  • Layout-Referenzen in Präsentationen
	  • Ressourcen-Referenzen in Layouts
	  • Präsentationsreferenzen im Nutzer-Profil
	  • Multimedia-Referenzen in Ressourcen

Nutzerprofil	• Einfügen einer Nutzer-Historie zur Identifikation von bisher benutzten Ressourcen durch einen Nutzer sowie der aktuell benutzten Präsentation • Erweiterung um Notifikationsdaten über Änderungen an Ressourcen, Layouts oder Präsentationen

Neben der zu ergänzenden Datenstruktur, die in einem nächsten Schritt um die oben genannten Elemente erweitert werden muss, sind Operationen zu implementieren, die direkt mit der Versionshaltung, der Dynamisierung der Referenzen und dem Benachrichtigungsmodell verbunden sind. Die Spezifikation dieser Operationen erfolgt in Kapitel 7.3.

7.2 CM- und DBMS-Versionierungsmodelle

Hier sollen bestehende Systeme untersucht werden und die Probleme und Mechanismen speziell im Hinblick auf verteilte multimediale Applikationen mit vielen Autoren betrachtet werden.

Das Versionsmangement von Objekten ist insbesondere in Softwareentwicklungsprojekten von essentieller Bedeutung. Zum einen unterliegen Softwaremodule vielen Änderungen, zum anderen arbeiten meistens mehrere Entwickler bzw. Teams gleichzeitig an Teilen eines Projektes. Daher ist gerade für die professionelle Entwicklung von Software ein Management der Konfigurationen von Softwaremodulen unumgänglich. Zur Unterstützung der Entwickler gibt es eine große Anzahl von weit entwickelten Configuration Managementsystemen (CM-Systeme), welche u. a. bekannte Methoden der Versionskontrolle einsetzen, um in Mehrbenutzerumgebungen die Verwaltung und Koordination von Designobjekten zu gewährleisten. Gerade weil diese Systeme eine Vielzahl von ausgereiften Funktionalitäten haben, bietet sich die Untersuchung dieser Systeme für diese Arbeit an. Daher werden in diesem Kapitel zunächst die diesen Systemen zugrundeliegenden Funktionalitäten vorgestellt. In einem zweiten Teil werden exemplarisch drei ausgewählte Systeme vorgestellt und miteinander verglichen.

Das Kernstück der Konsistenzkontrolle in CM-Systemen ist die Versionshaltung. Eine Datenbank stellt wiederum die benötigten Versionen der Designobjekte zur Verfügung. Wird

die Funktionalität von CM-Systemen mit Sammlung, Speicherung großer Mengen an reich strukturierter Daten kombiniert, sind auch entsprechende Mechanismen in Datenbankmanagementsystemen (DBMS) zu betrachten. Wie sich in Abschnitt 7.2.2 ergibt, lassen sich neben Elementen der Mehrbenutzerumgebung von CM-Systemen auch Konzepte von DBMS hier adaptieren. In diesem Abschnitt werden daher die typischen Versionierungskonzepte in Datenbanksystemen vorgestellt. Die Anwendbarkeit der erarbeiteten Konzepte wird abschließend in Abschnitt 7.2.3 evaluiert.

7.2.1 Versionierung in CM-Systemen

In [150] wurden die wichtigsten CM-Tools vorgestellt und kurz beschrieben. In [152] wurden Vergleichskriterien für CM-Tools untereinander entwickelt, anhand derer die Fähigkeiten dieser Tools eingeschätzt werden können, die für die zielgerichtete Entwicklung von komplexen Softwareentwicklungsprojekten notwendig sind. Diese Kriterien sind Language Support, Process Control, Versionierung/Baselining, Zugangskontrolle, Change Management, Reporting. Diese sechs Kriterien werden in [152] weiter durch vertiefende Fragen aufgegriffen und für jedes verfügbare CM-Tool beantwortet. Dabei wächst die Anzahl der Anforderungen auf angeblich derzeit 200 mögliche. Die sechs Evaluationskriterien lassen sich direkt unter den CM-Prozessen subsummieren, die in Abschnitt 3.3.1 beschrieben wurden. Für die Anwendbarkeit der Methoden auf multimediale Inhalte mit vielen Autoren bieten sich diese sechs Kriterien ebenfalls an. Diese einzelnen Kriterien und ihre Evaluierung für die oben beschriebenen drei CM-Tools bietet die nachfolgende Tabelle 9. Für eine Mehrbenutzerumgebung für viele Autoren können einige Elemente dieser CM-Funktionalitäten übernommen werden. Aufbauend auf den Szenarien in Kapitel 7.1.1 sind die benötigten Funktionalitäten den Funktionen der CM-Tools in Tabelle 10 zugeordnet und deren Relevanz erläutert. Dabei bezeichnet die Nummer die Funktionalität aus Tabelle 7. Kann ein CM-Tool nicht einer speziellen Funktion aus Tabelle 7 zugeordnet werden, wird eine Relevanz oder Irrelevanz für das Ziel dieses Abschnitts mit einem "+"- oder "-"-Zeichen gekennzeichnet.

7.2.2 Mechanismen in Datenbank-Systemen

Für Versionierung und Change Management sind für Datenbankmanagementsysteme bewährte Modelle und Algorithmen vorhanden. Deren Arbeitsweise wird nun in den nächsten Abschnitten beleuchtet und für die hier betrachtete Applikation adaptiert.

Die in Tabelle 6 und in Tabelle 8 geforderten Erweiterungen waren allgemein gehalten. Sie betrafen die Erweiterungen, um die in den Szenarien aus Kapitel 7.1.1 gemachten Forderungen zu erfüllen. Die in Abschnitt 3.3.2 dargestellten Versionierungssysteme stellen jedoch Bereiche heraus, die für DBMS schon breite Anwendung finden. Daher ist es in diesem Abschnitt erforderlich, die verbreiteten Methoden mit den Forderungen aus Kapitel 7.1.1 abzugleichen und gegebenenfalls diegroß gewünschten Funktionen und Erweiterungen weiter zu ergänzen.

Die Problembereiche, die in [25] diskutiert wurden, bedeuten im hier interessierenden Problemfeld:

- **Currency:** Auch für ein System, das multimediale Inhalte behandelt, ist die Forderung vorhanden, die aktuelle Version einer Ressource bestimmen zu können.

Tabelle 9: Drei CM-Tools im Überblick

CM-Prozess	Evaluationskriterium	Beschreibung	RCS	CVS	TC
Konstruktion	Versionierung / Baselining	• Versionshaltung für Designobjekte • Long Transaction • Konfigurationen	X	X	2
Team Arbeit	Zugangskontrolle	• Zugriffskontrolle auf Objekte aufgrund Arbeitsteilung • Verwaltung von Zugriffsrechten	-	1	2
Prozess Management	Change Management	• Vergabe von Entwicklungsstati für Objekte • Fortschrittskontrolle • Bearbeitungsrechte	-	-	2
Prozess Management	Process Control	• Entwicklungsprozess-definition • Einhaltung des Prozesses • Rollenvergabe in Prozessen	-	-	2
Prozess Management	Language Support	• Programmier-sprachenunterstützung • Verarbeitung der Programmstücke	-	-	2
Team Arbeit / Prozess Management	Reporting	• Generierung von Status-Reporten • Fortschrittsreports	-	-	2

Legende:
X - Vorhanden; **-** - Nicht vorhanden; **1** - Grundfunktionalität unterstützt; **2**- Erweiterte Funktionalität

- **Dynamische Konfigurationen:** In [25] wird ein Weg aufgezeigt, Konfigurationen dynamisch auszulesen, indem ein Suchpfad für bestimmte Versionen von Objekten festgelegt wird, die in die Konfigurationen einzufügen sind. Dynamische Konfigurationen finden ihre Entsprechung in dynamisch zusammengestellten multimedialen Präsentationen und in Hypertext-Beziehungen. Auch hier sind diese dynamischen Referenzen aufzulösen und durch die referenzierten Ressourcen zu ersetzen.
- **Workspaces:** Arbeitsbereiche werden durch die Nutzungsrechte bestimmt, die der Nutzer auf bestimmte Ressourcen setzt oder nicht. Das Rechtekonzept aus Kapitel 5 kann hier zur Einschränkung des Zugriffs verwendet werden, es können auch Gruppenrechte vergeben werden. Für das Versionierungsmodell ergibt sich hier die Anforderung, Präsentationen parallel und unabhängig voneinander bearbeiten zu können (dies bezieht sich nicht auf das gleichzeite Bearbeiten *monomedialer*, "atomarer", Inhalte, vgl. Abschnitt 2.3).

Tabelle 10: Relevante CM-Mechanismen

CM-Mechanismus	Entsprechende benötigte Funktionalitäten aus Tabelle 7	
Versionierung	Benötigt: Versionierung mit Versionsgraphen der Ressourcen, Präsentationen und Layouts	**1**
Zugangskontrolle	Zugangskontrolle wird durch die Mechanismen aus Kapitel 5 zur Verfügung gestellt. Als Vorgabe wird gleichzeitiger Zugriff auf eine Ressource durch Locking vermieden (s. auch Abschnitt 2.3).	+
Change Management	Ein Change-Management in größerem Umfang wird nicht benötigt, da nicht direkt die Entwicklung von Präsentationen das Ziel ist. Die Entwicklung der Präsentationen wird in dedizierten Werkzeugen außerhalb des Zielsystems geschehen. Fertige Präsentationen können dann gespeichert und gemeinsam genutzt werden. Jedoch ist bei Updates von Ressourcen die Benachrichtigung von anderen Nutzern, sowie die Koordination von zeitgleichen Updates (Concurrency Control) unabdingbar.	**2**
Process Control	Nicht betrachtet. Im Gegensatz zu technischer Entwicklung sind die Konsistenz- und Entwicklungszustände bei Lehrinhalten weder entsprechend klar definiert noch. Die Fortschrittskontrolle ist eher Gegenstand von Workflow-Mechanismen, die nicht im Rahmen dieser Arbeit betrachtet werden (s. Abschnitt Abschnitt 3.3.3).	**-**
Language Support	Diese für CM-Tools wichtige Eigenschaft wird hier nicht gebraucht: das vorgeschlagene System soll von der Repräsentation der Inhalte vollkommen abstrahieren, wogegen CM-Tools die verwendete Programmiersprache einer Ressource betrachten können.	**-**
Reporting	Erstellung von Fortschritts- und Statusreportenwird hier (ebenso wie Process Control, s. Abschnitt 3.3.3) nicht betrachtet.	**-**

- **Logische und physische Repräsentation:** Die physische Repräsentation der Ressourcen ist zwar ein relevanter Problembereich, jedoch nicht Gegenstand dieser Arbeit. Hier soll die notwendige Applikationssemantik und die dafür notwendigen Strukturen identifiziert werden.
- **Change/Constraint Propagation:** Hier werden die Änderugen an einem Objekt wiederum an andere Objekte propagiert, da diese miteinander verknüpft sind. Die Übernahme dieses Mechanismus bietet sich an, da die Daten, welche versioniert werden, ebenfalls hierarchisch verknüpft sind. Da keine Äquivalenzbeziehungen zwischen Ressourcen bestehen, wird dieser Teil nicht übernommen.
- **Vererbung:** Dieser Problembereich wird in dieser Arbeit noch nicht betrachtet, vielmehr wurde eine Datenstruktur ohne Vererbung gewählt (Abschnitt 4.3).

Dieses Modell der Versionierung in einer objektorientierten Datenbank benötigt Erweiterungen der Datenstruktur der Objekte, um diese Strukturen zu realisieren. Erweiterungen dieser

Art wurden schon in [26] diskutiert. Zur Versionierung von Multimedia-Daten ist es notwendig, die Datenstruktur analog zu erweitern. Dies sind die Elemente:

- Object-ID der Version zur eindeutigen Identifikation der Version einer versionierbaren Ressource.
- Version Number zur Einordnung in den Versionsbaum und der Derivation der Versionen.
- Zeiger auf Nachfolger- bzw. Vergängerversion eines versionierten Objektes.
- R/L/E Indicator: Dieses Attribut ist wichtig für die Differenzbildung von Nachfolgeversionen, ist also eine implementierungsbezogene Komponente, die eher die physikalische Speicherung und Abfrage der Versionen betrifft, so dass dieses Element für den Rahmen dieser Arbeit nicht relevant ist.
- State: Dieses Attribut ist für die Implementierung eines Locking relevant, um konkurrierende Änderungszugriffe auszuschließen. Dieser Lock-Status dient zur Sequenzialisierung solcher konkurrierenden Zugriffe.

Ebenso wird ein Notifikationssystem analog zu [37] benötigt, welches dafür sorgt, dass bei Aktualisierungen die Nutzer der aktualisierten Objekte in Kenntnis gesetzt werden.

Die Bearbeitung der Ressourcen muss parallel erfolgen können, da die Nutzer unabhängig voneinander Ihre Präsentationen an unterschiedlichen Bereichen bearbeiten. Ein Merging der bearbeiteten Versionen ist nicht vorgesehen. Somit müssen auch Seitenäste im Gegensatz zu [37] ermöglicht werden. Dies hat primär Auswirkungen auf das Currency-Problem, da es nun aufgrund der Seitenäste keine eindeutige letzte Version auf dem Revisionspfad gibt. Die Diskussion dieser Probleme und gesamtheitliche Lösung geschieht in Abschnitt 7.3.

Die Ergebnisse dieses Teilabschnitts sind nochmals zur Übersicht in der Tabelle 11 zusammengefasst, wobei irrelevante Bereiche herausgelassen wurden. Dabei werden die relevanten Problembereiche der Versionierung in Datenbanken den gewünschten Funktionalitäten zugeordnet. Ist keine Zuordnung möglich, so ist das entsprechende Feld leer.

Tabelle 11: Relevante Mechanismen von DBMS

DBMS-Mechanismen	Entsprechende benötigte Funktionalität aus Tabelle 7	
Currency	Currency Control muss gewährleistet sein. Siehe hierzu auch Tabelle 6 in Kapitel 7.1.1.	1,3,6
Change Propagation	Änderungen der betroffenen Objekte müssen ebenso durchgeführt werden, da diese neben hierarchischer Verknüpfung auch durch Referenzen verknüpft sind.	1,3
Workspaces	Diese finden Ihre Realisierung in der Vergabe von unterschiedlichen Zugriffsrechten sowohl für einzelne Nutzer als auch für Gruppen.	7
Dynamische Konfigurationen	Präsentationen mit aktuellem Inhalt werden erzeugt mit dynamischen Referenzen auf Ressourcen und Layouts .	1,3,6
Bearbeitungsart	Parallele, konkurrierende Bearbeitung einer Version einer Ressource wird hier nicht betrachtet. Parallele Bearbeitung ergibt hier immer verschiedene Varianten der bearbeiteten Ressource.	

Tabelle 11: Relevante Mechanismen von DBMS

DBMS-Mechanismen	Entsprechende benötigte Funktionalität aus Tabelle 7	
Erweiterungen der Datenstruktur	Grundlegende Erweiterungen zur bisherigen Datenstruktur sind die Attribute Versionnumber, Pointer to parent/child version, Version State. Desweiteren sind Erweiterungen im Nutzerprofil vorzunehmen. Siehe hierzu auch Tabelle 8.	1,6,7
Notifikation	Die Notifikation der Nutzer nach Änderungen geschieht nach dem Modell in Tabelle 5.	2

7.2.3 Relevanz der Mechanismen

Neben den Zielen einer Versionsschnittstelle (Abschnitt 7.1.1) ergeben sich aus den im vorhergehenden Abschnitt diskutierten Modellen Konsequenzen bezüglich der Realisierung. Für eine Mehrbenutzerumgebung, wo Benutzer auf die versionierten Daten dynamisch zugreifen und Änderungen vornehmen wollen, müssen die Zugriffe über eine Schnittstelle abgewickelt werden, welche derart spezifiziert wird, so dass die Datenstruktur konsistent bleibt. Die Schnittstelle muss im Wesentlichen zwei Operationen durchführen: Lese- und Schreiboperationen. Bei Leseoperationen muss die Schnittstelle bei Angabe einer Object-ID oder Object-ID mit Versionsnummer immer die entsprechende Ressource ausgeben. Bei Updates müssen die Operationen zur Change Propagation und der Notifikation durchgeführt werden.

Bei der Untersuchung der verschiedenen Versionskontrollsysteme in diesem Kapitel hat sich gezeigt, dass durchaus vorhandene Konzepte übernommen werden können. Durch die Nutzung dynamischer Referenzen ergeben sich besondere Anforderungen hinsichtlich der Change Propagation und der Notifikation von Anwendern. Ein Notifikationsmodell ergab sich schon direkt aus den Nutzungsszenarien in Abschnitt 7.1.1, für welches weitgehend entsprechende Mechanismen aus DBMS verwendet werden können. Aufbauend auf diesen Anforderungen an ein Versionskontrollsystem kann nun in einem letzten Schritt das System und darauf folgend die Versionssschnittstelle spezifiziert werden. Die Ergebnisse des Kapitels sind in Tabelle 12 zusammengefasst.

7.3 Spezifikation des Ziel-Versionierungssystems

Die benötigten Erweiterungen über die Datenstruktur aus Abschnitt 4.3 werden in diesem Kapitel festgelegt, die Einbettung dieser Erweiterungen im bestehenden Datenbankschema erläutert und die Methoden und Algorithmen zur Verarbeitung dieser Versionierungsdaten entwickelt und beschrieben.

Diese erweiterte Datenstruktur bildet das informatorische Grundgerüst, auf das die Operationen der Versionierung aufbauen werden. Es wird eine prototypische Implementierung der zentralen Funktionen dieses Versionierungssystems vorgestellt. Ziel dieses Abschnitts ist es die aufgezeigten Anforderungen genauer zu untersuchen und zu spezifizieren.

Tabelle 12: Relevante Mechanismen: CM und DBMS kumuliert

Funktionalität (vrgl. Tabelle 7)		CM-Modell	DBMS-Modell	Erläuterungen
1	Versionsbäume für Ressourcen, Präsentationen und Layouts	• Versionierung	• Currency • Dynamische Konfigurationen • Erweiterungen der Datenstruktur	• Benötigt: Versionierung mit Versionsgraphen der Ressourcen, Präsentationen und Layouts • Currency Control wird benötigt (Tabelle 6 in Abschnitt 7.1.1) • Benötigt: dynamische Referenzen • Erweiterungen der Datenstruktur: Version Number, Predecessor/Successor Versions, Version State, Erweiterungen isind m Nutzerprofil vorzunehmen. (Tabelle 8)
2	Nutzernotifikation bei entfernten Updates	• Change Management	• Notifikation	• Dem Nutzer muss eine neue Version einer Ressource und neue Versionen referenzierter Ressourcen signalisiert werden können. • Konfiguration der Art und Häufigkeit der Notifikation: Änderungen eines Ressourcen-Unterbaums sind zeitweise auch unkritisch und erwartet.
3	Aktualisierung der Präsentationen		• Currency • Change Propagation • Dynamische Konfigurationen	• Change Management: neue Versionen nur entlang der *enthalten*-Hierarchie. Weitere Beziehungen (Referenzen) werden nur durch Benachrichtigung anderer Nutzer über Updates berücksichtigt (s. Tabelle 5). • Currency Control muss gewährleistet sein (Tabelle 6).
4	Ein- und Ausschalten der Aktualisierung			• In kritischen Bereichen will der Nutzer selbst nach einer Notifikation entscheiden, ob die aktuellere Version genutzt werden soll.
5	Kontrollmodus			• Eine Übersicht darüber, wo neuere Versionen verfügbar sind (Szenario 6)
6	Konfigurierbare Referenzen		• Currency • Change Propagation • Dynamische Konfigurationen • Erweiterungen der Datenstruktur	• (s.o., Erläuterungen zu Versionsbäumen)
7	Nutzer-Kontext		• Workspaces • Erweiterungen der Datenstruktur	• Der Nutzerkontext muss Fokus und Historie (jeweils mit Versionsinformation) der Arbeit des Nutzers speichern.

7.3.1 Versionsbaum für Ressourcen, Präsentationen und Layouts

Zur Verwirklichung einer Versionierung von Datenelementen ist es erforderlich, die versionierbaren Elemente der Datenstruktur mit Versionsattributen zu versehen. Diese sind Ressourcen, Präsentationen, Layouts und Referenzen. In diesem Kapitel werden zunächst die drei Erstgenannten behandelt.

Jeder Ressourcenknoten enthält fortan einen Versionsknoten *version* mit folgend aufgelisteten Attributen:

- ***id***: Gibt die Versionsnummer an und hat als Initialwert immer den Wert “1.0”, was für die Ausgangsversion steht.
- ***date***: Gibt das Datum an, an dem diese Ressource erstellt wurde.
- ***creator***: Gibt den Nutzer an, der diese Version der Ressource erstellt hat.
- ***lockstate***: Gibt Zugriffsstatus der Ressource an. Der Wert “unlocked” bedeutet uneingeschränkten Schreib- und Lesezugriff, “nowrite” steht für Schreibverbot auf Ressource und “noread” steht für Leseverbot für die Ressource. Standardmäß ist hier der Wert des Attributs auf “unlocked” gesetzt.
- ***lockeddate***: Gibt an, zu welchem Datum diese Ressource gesperrt wurde, d. h. wann version.lockstate auf einen anderen Wert als “unlocked” gesetzt wurde.
- ***lockedby***: Gibt bei Einschränkung des Zugriffs auf diese Ressource an, welcher Nutzer diese Ressource gesperrt hat.

Jede Ressource kann mehrere Versionsnachfolger haben, jedoch immer nur einen Versionsvorgänger. Darum enthält das Element *version* keine oder mehrere Elemente *version.successor* und kein oder nur ein Element *version.predesessor*. Die Ressource, welche keinen Vorgänger hat, ist auch die Basisversion, von der alle nachfolgenden Versionen abgeleitet werden. Die Vorgänger- und Nachfolgerknoten haben Attribute, die auf die entsprechenden Nachbarknoten verweisen. Dabei genügt die Angabe der Versionsnummer im Attribut *referenced.version* sowie der Ressourcen-ID im Attribut *referenced.id*. Somit hat die Datenstruktur der Ressourcen jetzt neben der vertikalen Part-of-Beziehungen auch eine dazu orthogonale Struktur mit Vorgänger- und Nachfolgerbeziehungen zwischen Ressourcenversionen, wobei die jeweils neu erzeugten Versionen in der entsprechenden Ressourcenliste abgelegt werden.

Für das in Kapitel 7 geforderte Rechte-Management müssen die Versionsknoten auch Rechteinformationen enthalten. Hierzu ist der Ersteller der Version *(version.creator*), das Erstelldatum (*version.date*), der Zugriffstatus (*version.lockstate*), Datum der Zugriffsperrung (*version.lockeddate*) sowie der Sperrer dieser Ressource (*version.lockedby*) anzugeben. Das zentrale Attribut zur Realisierung eines Locking-Mechanismus ist *version.lockstate*, welches einen möglichen gleichzeitigen Schreibzugriff durch mehrere Nutzer verhindert. Dieses Attribut dient auch der Bearbeitungsserialisierung von Ressourcen, so dass bei Wunsch eine arbeitsteilige Bearbeitung der Ressourcen ermöglicht wird.

Neben den Ressourcen enthält die vorgeschlagene Datenstruktur auch Präsentationen (Abschnitt 4.3). Die Präsentationen sind ähnlich den Ressourcen hierarchisch strukturiert und referenzieren die jeweils benutzten Ressourcen. Ebenso wird in diesen Präsentationen auf Layouts aus der *layouts.list* referenziert. Eine Präsentation ist somit mit der Angabe der verwende-

ten Ressource und dem verwendeten Layout komplett. Die Datenstruktur der Präsentationen wird in gleicher Weise wie die der Ressourcen um einen Versionsknoten erweitert, womit auch die Präsentationen in eine horizontale Versionierungsstruktur und eine vertikale Part-of-Relationen-Struktur gebracht werden. Die erweiterte Datenstruktur für Ressourcen, Präsentationen und Layouts ist in Abbildung 23 dargestellt, wobei der Vollständigkeit halber auf Abschnitt 7.3.4 vorgeifend auch das *pseudo-part-of*-Element eingezeichnet wurde.

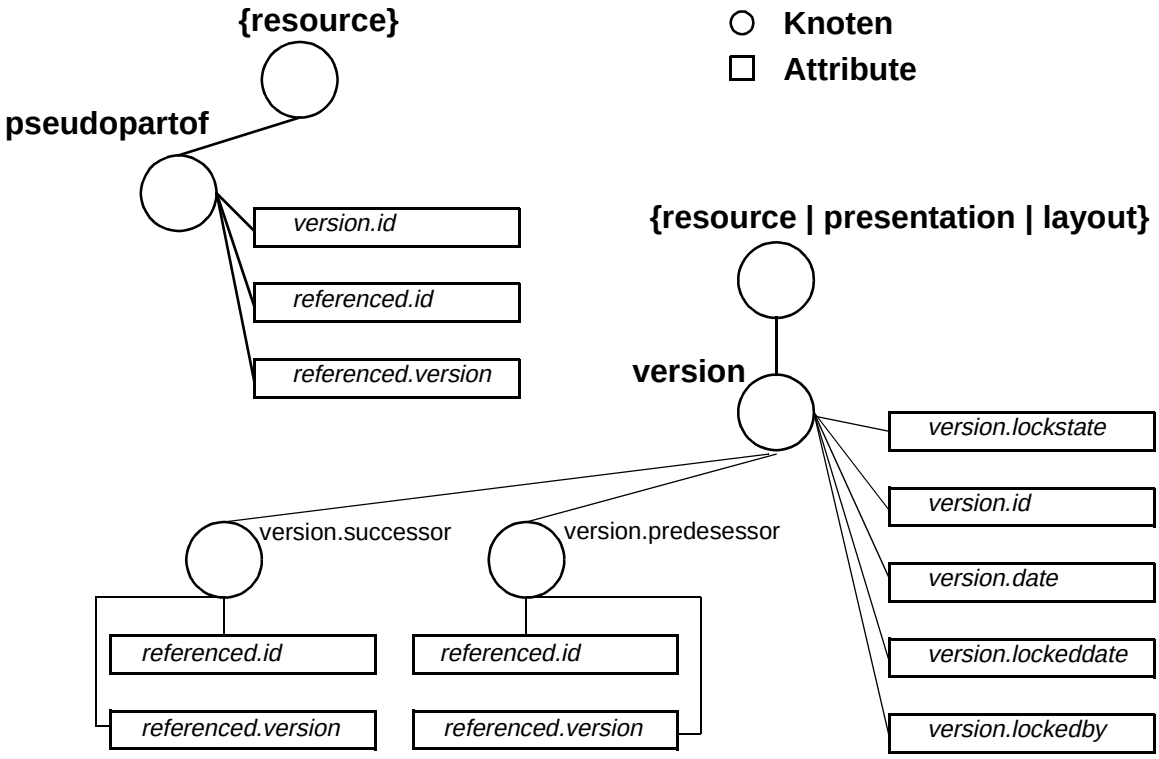

Abbildung 23: Datenstrukturerweiterungen in Ressourcen- Präsentations- und Layoutknoten

Da Präsentationen, Layouts und Ressourcen jeweils eine Versionsstruktur haben, müssen aufgrund der Referenzierungen untereinander auch die Referenzen um Versionsinformationen erweitert werden. Dies erfolgt im nächsten Abschnitt.

7.3.2 Änderung der Datenstruktur von Referenzen

Für die Zusammenstellung einer Präsentation sind mit dem Ansatz aus Kapitel 4.2 zwei Elemente notwendig: die zu präsentierenden Ressourcen und das anzuwendende Format. Für die Formatierung können Layouts im System (*layouts.list*) abgelegt werden. Die Präsentationen verweisen auf die entsprechenden Ressourcen-Unterbäume und das benutzte Layout.

In der Datenstruktur in Abschnitt 4.3 (ohne Versionsinformationen) wurde diese Funktionalität dadurch erreicht, dass in den Attributen vom Knoten *presentation* einmal die Ressource per *resource.ref* und das Format per *layout.ref* referenziert wurde. Da nun diese Referenzen zusätzlich Angaben über die Version und den Typ der Referenz haben müssen, muss hier ebenfalls eine Erweiterung der Datenstruktur vorgenommen werden.

Dies wird dadurch realisiert, dass die Attribute nun in Knoten umgewandelt werden, die auf die entsprechenden Ressourcen oder Layouts referenzieren: *resource.ref* und *layout.ref*. Diese Knoten erhalten folgende Attribute:

- ***type***: Typ der Referenz “current” oder “frozen” für statische oder dynamische Referenz
- ***id***: ID-Nummer der referenzierten Ressource oder Layout
- ***version***: Versionsnummer des referenzierten Knotens

Ebenso werden die Referenzen in den Layouts und im Nutzerkontext in Knoten umgewandelt und mit diesen Attributen versehen.

Die Multimediareferenzen in den Ressourcen, welche bisher auf eine Ressource mit einem Substitution-String verwiesen, sind in Zukunft auch als Knoten *multimedia.reference* in den Ressourcen realisiert. Diese Knoten enthalten als Referenz wieder ein *resource.ref*-Element, welches per *referenced.id* und *referenced.version* auf eine Version einer Ressource dynamisch oder statisch (*reference.type*) verweist. Der Substitution String ist hier im Attribut *substitution.string* enthalten.

7.3.3 Erweiterung des Nutzerprofils

Im Nutzerprofil ohne Versionierungserweiterungen sind Verweise, Referenzen, auf die letzten bearbeiteten oder genutzten Inhalte enthalten, um dem Nutzer eine unkomplizierte Fortführung seiner Arbeit zu erlauben.

Neben einer simplen Erweiterung um die Versionsidentifikation des Arbeitsfokus ist im Nutzerkontext auch noch die folgende Entscheidung des Nutzers festzuhalten. Genau wie bei der Nutzung (Referenzierung) von Ressourcen sind auch hier zwei Alternativen gegeben: der Nutzer arbeitet mit der aktuellen Version oder mit einer bestimmten Version eines Inhaltes. Dies muss das System der Interaktion mit dem Nutzer entnehmen können und in der Kontextinformation vermerken.

Die Nutzerhistorie wird in dem Knoten *user.history* gespeichert. Diese Nutzerhistorie enthält alle Präsentationen in einer Liste, die der Nutzer bis dato benutzt hat. Der Knoten *user.context* enthält die dynamische oder statische Referenz auf eine Präsentation, da nun Präsentationen auch versioniert vorliegen. Der Nutzer hat hier wie gehabt die Möglichkeit, auf die aktuelle oder eine bestimmte Präsentation zu verweisen.

Der Knoten *user.context* ist nicht nur in der *user.history* enthalten, sondern auch in dem Knoten *user*, wodurch die vom Nutzer gerade benutzte Präsentation verdeutlicht wird. Derselbe Knoten findet sich auch in der *user.history* wieder. Damit eine Reihenfolge der benutzten Nutzer Kontexte hergestellt werden kann, werden analog den Versionsnummern fortlaufende Nummern im Attribut *number* vergeben. Somit kann die Datenstruktur, welche für die Versionierung interessant ist, in folgender Abbildung 24 zusammengefasst werden.

Desweiteren ist die Notifikation von Nutzern bei entfernten Updates vorgesehen (siehe in Abschnitt 7.1.1, Tabelle 5). Um diese zu erreichen, müssen die Benachrichtigungen auch in Abwesenheit des Nutzers gespeichert sein. Dies wird dadurch erreicht, dass sämtliche Benachrichtigungseinstellungen und Nachrichten in einem Element *notification* des *user*-Knotens abgelegt werden. Dieser Knoten enthält zunächst ein Attribut *notification.type*, worin die Art

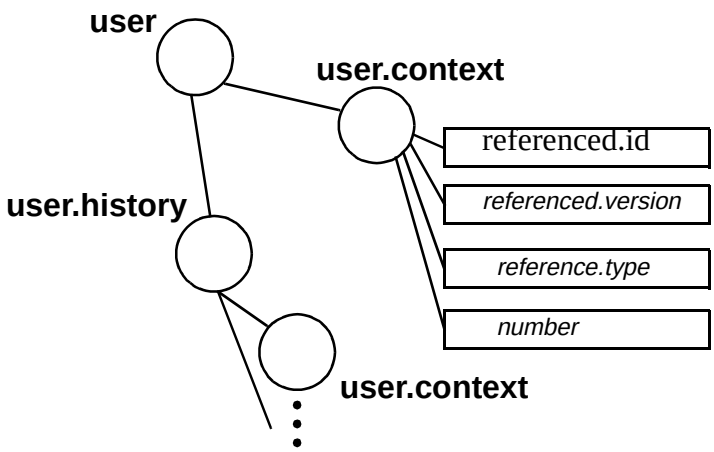

Abbildung 24: Nutzerprofil für Historie der benutzten Präsentationen

der Notifikation (push oder pull, siehe Tabelle 5) eingestellt werden kann. Gleichzeitig erhält es auch noch ein Attribut *notification.form*, was anzeigt, welche Form der Benachrichtigung gerade erwünscht ist. Anhand von *notification.type* kann angegeben werden, in welcher Form die Benachrichtigung stattfinden soll. Dies kann entweder E-mail oder Aktualisierungsindizierung in Form von Flags sein. Das Attribut *recent.updates* im Knoten *notification* zeigt dem Nutzer an, dass Änderungen an den vom Nutzer benutzten Präsentationen stattgefunden haben. Die Aktualisierungsnachrichten werden der Liste *updates.list* gespeichert, welche alle Notifikationen in den Knoten *update* enthält. Diese Knoten haben als Attribut die Identifikationsnummer der geänderten Ressource *resource.id*, die ursprüngliche Versionsnummer *version.id*, sowie die neue Versionsnummer *newversion.id*. Daneben ist noch der Nutzer, welcher die Änderung vorgenommen hat und im Attribut *update.by* sowie in *update.date* das Datum der Änderung enthalten. Ein Kommentar bezüglich der Änderung kann ebenfalls in den Knoten *update* eingefügt werden.

Die resultierende Datenstruktur zeigt Abbildung 25. Die Kapselung der Operationen und Einstellungen zur Notifikation werden im Kapitel 7.3.4 vorgestellt.

7.3.4 Spezifizierung der Versionsschnittstelle

Die Versionsschnittstelle bildet das Bindeglied zwischen einer privaten Datenbank, in der die Inhalte ohne Versionierung abgelegt sind, und der öffentlichen Datenbank, in der die Versionsinformationen enthalten sind. Hierdurch wird eine Kapselung von Operationen an der Datenbank ermöglicht, so dass eventuelle Inkonsistenzen vermieden werden. Die Aufgaben der Schnittstelle bestimmen sich aus den Anforderungen, die an die Versionierung in Abschnitt 7.1.1 gestellt wurden. Die Aufgaben der Schnittstelle lassen sich in zwei Aufgabengebiete aufteilen:

- **Lese-Operationen:**
  Bei Angabe von bestimmten Parametern werden an die anfragende Applikation Daten aus der öffentlichen Datenbank bereitgestellt.

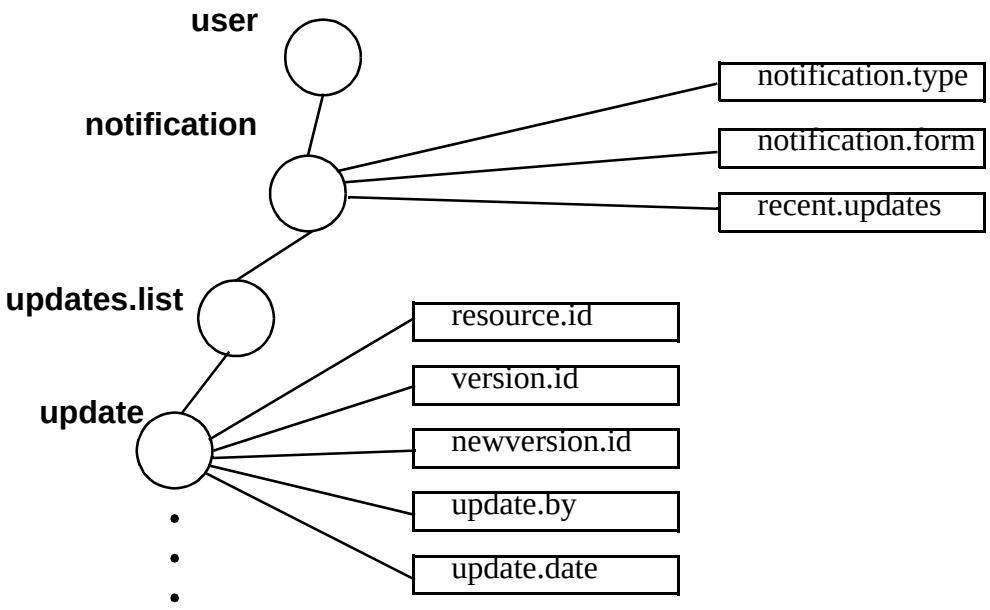

Abbildung 25: Notifikationsdatenstruktur im Nutzerprofil

- **Schreib-Operationen:**
  Hier wird an die Schnittstelle ein bestimmter Schreibbefehl (z. B.: create, change, delete, insert) mit den entsprechenden Parametern übergeben, so dass diese alle zugehörigen Schreibbefehle mit Versionsupdates, Referenzupdates und Nutzernotifikationen durchführen kann.

Im einzelnen muss die Schnittstelle die im folgenden aufgelisteten Aufgaben erfüllen, damit die zugreifende Applikation diese auswerten kann.

- Aktualisierung der Versionsbäume: Diese Aufgabe hängt mit der Change Propagation bei Update eines Knotens zusammen.
- Aktualisierung der dynamischen Referenzen bei Änderungen der Versionsbäume: Hier müssen bei Änderungen an bestimmten Teilen der Datenbank (Ressourcen, Präsentationen, Layouts) die dynamischen Referenzen auf diese aktualisiert werden. Hier ergeben sich zusätzliche Aufgaben, die sich bei Erstellung von Seitenästen ergeben. Siehe hierzu die weiteren Ausführungen.
- Auslesen von Inhalten: Hier müssen bei Änderungen an bestimmten Teilen der Datenbank (Ressourcen, Präsentationen, Layouts) die dynamischen Referenzen auf diese aktualisiert werden.
- Lesen von Metadaten über die Inhalte: Dies ist relevant für den Kontrollmodus, wo Abfragen zu Versionsinformationen gestartet werden.
- Änderungen an den Nutzereinstellungen: Änderungsbefehle des Nutzers an seinen Einstellungen müssen auch über diese Schnittstelle laufen.
- Nutzerinteraktion: Benachrichtigung bei entfernten Updates, Änderung der Nutzerprofile.

Diese Aufgaben werden nun näher besprochen.

Versionsaktualisierungen in der öffentlichen Datenbank

Durch die hierarchische *part-of*-Relation der Ressourcenknoten sind bei Änderungen nicht nur die Knoten selbst, sondern auch deren Vaterknoten betroffen. Das bedeutet, dass für beliebige Änderungen innerhalb eines Baumes immer die Ressource verändert wird, die alle Ressourcen enthält. Dies ist in allen Fällen der Wurzelknoten. Man stelle sich zur Illustratation ein Dokument vor, welches mehrere Kapitel enthält. Diese wiederum enthalten Text, Grafiken etc. Nun wird ein Text in einem Abschnitt verändert. Dies hat nicht nur zur Folge, dass sich der Abschnitt verändert hat, sondern auch das Dokument insgesamt, da dieser Text in dem Dokument enthalten ist.

Der Versionsbaum für Ressourcen wird allerdings nicht nur die Wurzelknoten mit Versionsinformation versehen, sondern auch die darin enthaltenen veränderten Ressourcen. Dies wird dadurch erreicht, dass jedem veränderten Knoten ein Versionsnachfolger *version.successor* zugeordnet wird. Dies geschieht ebenfalls mit allen Knoten, in denen diese Ressource enthalten ist, bis hin zum Wurzelknoten, da diese implizit auch verändert wurden. Da die Enthaltungsrelationen auch für die neuen Versionen der Knoten gelten, wird zur Erzeugung der neuen Version des Ressourcenbaumes der Pfad bis zum Wurzelknoten herauskopiert. Dies entspricht der Change Propagation in Anlehnung an [25] in Abbildung 7. Alle anderen Knoten, die nicht auf diesem Pfad liegen, werden in der neuen Version des Ressourcenbaumes referenziert. Die Referenzierung geschieht aus dem Grund, dass zwar diese Ressourcen in den neuen Versionen der veränderten Ressourcen enthalten sind, jedoch sich selbst nicht verändert haben. Sie werden *pseudo-part-of*-Elemente genannt, da beim Auslesen an Stelle dieser Elemente die durch sie referenzierte Ressource eingefügt wird. Diese *pseudo-part-of*-Elemente sind im Gegensatz zu den herkömmlichen statischen oder dynamischen Referenzen immer statisch und können nicht dynamisiert werden. Durch diese Referenzierung wird vermieden, dass unveränderte Ressourcen nicht mehrfach vorkommen. Somit ergibt sich für die Versionsschnittstelle zwingend die Anforderung, bei einer Leseoperation alle *pseudo-part-of*-Elemente durch die entsprechende Ressource zu ersetzen. Werden mehrere Ressourcen verändert, so geschieht dasselbe analog für mehrere Ressourcen. Die Pfade bis zum Wurzelknoten, auf denen diese Ressourcen liegen, werden in den neuen Ressourcenbaum übernommen. Alle anderen Ressourcen werden durch *pseudo-part-of*-Elemente referenziert. Da sich der Inhalt der Ressourcen verändert hat, werden diesen im neuen Ressourcenbaum neue ID-Nummern vergeben. Gleichzeitig haben diese nun im Versionsbaum einen Vorgänger und enthalten somit als Versionsinformation einen zusätzlichen Knoten *version.predesessor.*

Da in den Ressourcen auch Multimediareferenzen vorhanden sein werden und diese nichts weiter darstellen als statische oder dynamische Referenzen, werden diese veränderten Referenzen auch komplett in den neuen Ressourcenbaum kopiert, da eine Referenzierung durch ein *pseudo-part-of*-Element wenig Sinn ergeben würde. Bei Änderung einer Multimediareferenz in einem Ressourcenknoten wird der Ressourcenknoten verändert und somit behandelt, wie eine Ressourcenänderung, d.h. der veränderte Knoten wird in den neuen Baum kopiert.

Die Funktionsweise der Updates veranschaulicht Abbildung 26.

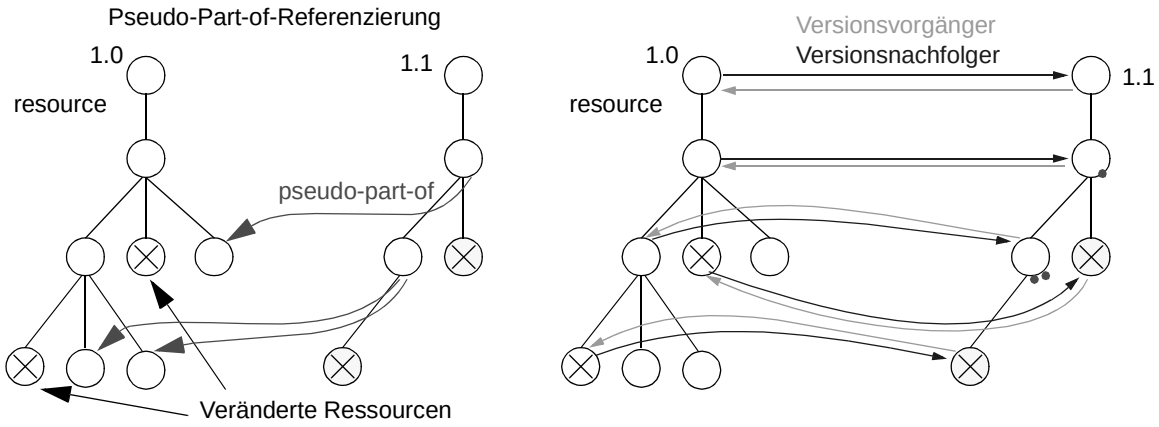

Abbildung 26: Gewählter Update Mechanismus

Sind jedoch Aktualisierungen an Ressourcen gewünscht, welche schon *Pseudo-part-of*-Referenzen haben, so stellt sich die Frage, wie der Update-Mechanismus nun funktionieren muss, damit eine Change-Propagation bis zum Wurzelknoten noch durchgeführt werden kann.

Dazu ist es notwendig zu berücksichtigen, dass beim Auslesen von Ressourcenbäumen in die private Datenbank die *pseudo-part-of*-Elemente aufgelöst und durch die entsprechenden Ressourcen ersetzt werden. Somit findet das Kopieren der Knoten bis zum Wurzelknoten schon in der privaten Datenbank statt, so dass die Schnittstelle die Version des Baums nur noch in die Ressourcenliste in der öffentlichen Datenbank einzufügen und die Informationen zu Versionsnachfolgern und -vorgängern zu ergänzen hat. Da die älteren Versionen der Knoten durch ihre Object-ID identifiziert werden, werden dort die entsprechenden Einträge für Versionsnachfolger eingefügt. Das in Bezug auf Abbildung 26 modifizierte Schema der Updates zeigt Abbildung 27, wobei dort beispielhaft ein mit *pseudo-part-of*-Elementen versehener Baum ausgelesen, aktualisiert und wieder eingespeichert wird.

Der Update-Mechanismus für Präsentationen, die auf Ressourcen und Layouts verweisen, geschieht etwas anders. Da Präsentationsknoten immer je eine Referenz auf ein Layout und eine Ressource haben, wird bei Änderung eines der Präsentationsknoten der komplette Baum als neue Version mit den geänderten Referenzen in die Präsentationsliste *presentations.list* eingefügt, da die Referenzierung von Präsentationsknoten, die nur Referenzen enthalten, keinen Sinn ergeben würde. Somit wird das *Pseudo-part-of*-Element für die Präsentationen nicht benötigt. Bei den Layout-Bäumen wird auch auf die Verwendung von *pseudo-part-of*-Elementen verzichtet.

Dieses hier vorgestellte Modell der Aktualisierungspropagation darf jedoch an dieser Stelle nicht enden. Zu berücksichtigen sind noch die dynamischen Referenzen, deren spezifizierte Aufgabe es ist, immer auf die aktuelle Version eines Knotens zu verweisen. Um dieses Currency-Problem in den Griff zu bekommen, wird im nächsten Abschnitt diskutiert, welche Version zunächst als aktuelle Version gilt und wie diese ermittelt wird. Dann bleibt der

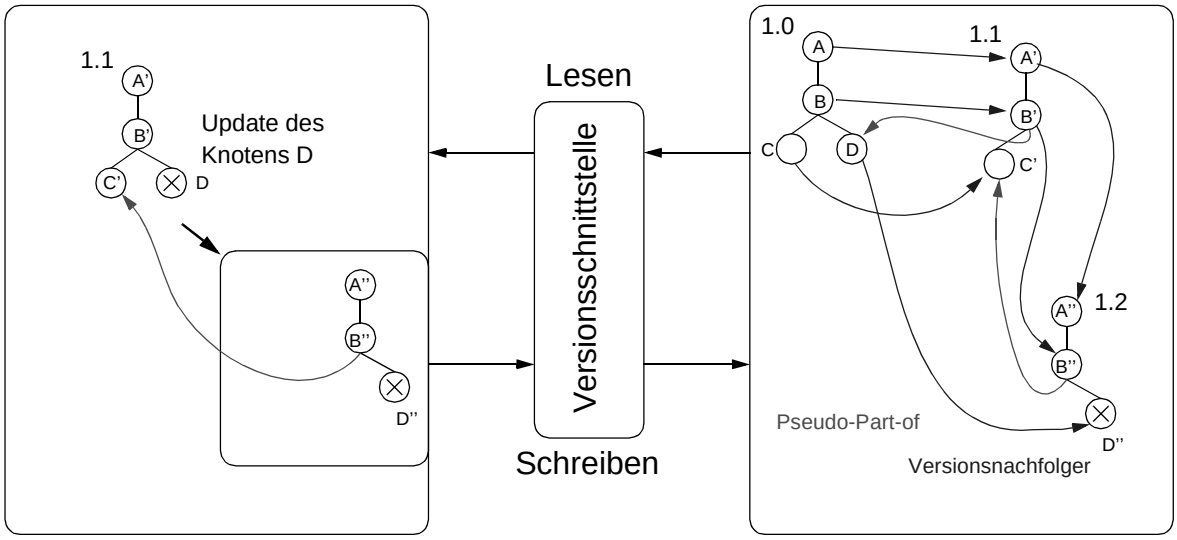

Abbildung 27: Update der Knoten über die Versionsschnittstelle

Versionsschnittstelle nur noch, nach Aktualisierungen an Knoten die dynamischen Referenzen auf die aktuelle Version zu setzen.

Aktualisierung der Referenzen

Neben der Change Propagation innerhalb der Bäume müssen bei Versionsupdates auch die dynamischen Referenzen auf die jeweils aktuelle Version einer Ressource, Präsentation oder Layout (im folgenden als Knoten bezeichnet) gesetzt werden. Diese Referenzen können sein:

- Ressourcen- und Layout-Referenzen in Präsentationen
- Ressourcen-Referenzen in Layouts
- Präsentationsreferenzen im Nutzer-Profil
- Multimedia-Referenzen in Ressourcen

Die Aktualisierung für diese Referenzen soll im folgenden beschrieben werden. Dabei ist es die grundlegende Idee, dass Aktualisierungen an den dynamischen Referenzen, die das Versionierungssystem von sich aus durchführt, keine neue Version des Inhalts darstellen. Vielmehr sollen nur Änderungen, die die Nutzer vornehmen, eine Änderung der Versionen der Knoten bewirken, die diese Referenzen enthalten.

Nach den Szenarien aus Abschnitt 7.1.1 ist davon auszugehen, dass die Nutzer oft die aktuelle Version einer Ressource benutzen werden. Daher werden viele Aktualisierungen an Knoten immer an der bis dahin aktuellen Version dieses Knotens stattfinden. Die aktuelle Version ist jeweils die Version in der Datenbank, die keine Versionsnachfolger besitzt.

Es kann weiterhin sein, dass Nutzer ältere Versionen eines Knoten benutzen und somit bei Aktualisierungen dieser Knoten einen neuen Seitenast erhält. Für den Nutzer dieses Stranges ist die aktuelle Version eines Knotens eine andere als für den Nutzer der Versionen auf dem Hauptstrang. Diesen Umstand veranschaulicht Abbildung 28.

Daher muss die Versionsschnittstelle so konzipiert werden, dass dieser Umstand bei der Aktualisierung der Referenzen berücksichtigt wird: wenn mehrere neue Varianten zur Aktualisierung

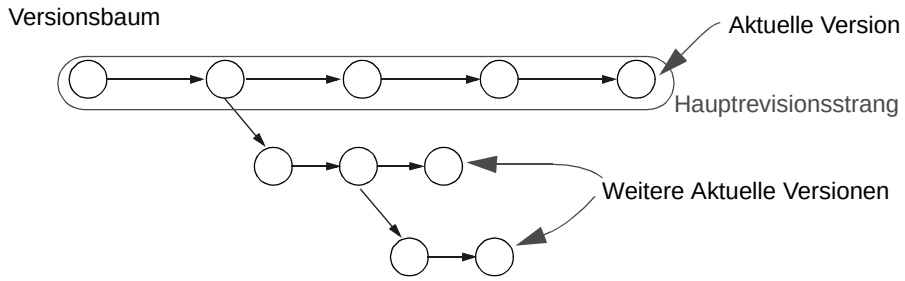

Abbildung 28: Currency

verfügbar sind, kann das System nur durch Interaktion mit dem Nutzer entscheiden, welche Variante die für die jeweilige Referenz aktuelle ist. Dieses trifft für Referenzen innerhalb von Ressourcen und Präsentationen genauso wie in Nutzerkontexten zu.

Ressourcenknoten können auf andere Ressourcenknoten verweisen, Layouts verweisen wiederum auch auf Ressourcen. Sind diese Referenzen statisch, so zeigen diese Referenzen immer auf eine Ressource mit bestimmter Versionsnummer. Diese Referenzen werden vom Versionierungssystem nicht angetastet. Anders ist es bei den dynamischen Referenzen, die vom Typ "current" sind. Diese dienen dazu, immer die aktuelle Ressource zu referenzieren.

Wird also eine Ressource aktualisiert und der neue Ressourcenbaum in die entsprechende Ressourcenliste eingefügt, so müssen gleichzeitig alle Referenzen in den Layout- und Ressourcenbäumen vom Typ "current" auf die aktuelle Version gesetzt werden, d.h. auf den direkten Versionsnachfolger des aktualisierten Knotens. Dabei ist zu berücksichtigen, dass die Aktualisierung einer dynamischen Referenz nicht einer neuen Version des Ressourcenbaumes entspricht, in der diese Referenz enthalten ist.

Wird die letzte Version auf einem Revisionsstrang aktualisiert, so genügt es, dass die Versionsschnittstelle neben der Aktualisierung des Versionsbaumes auch alle dynamischen Referenzen auf die neue Version setzt. So ist für die meisten Nutzer die Aktualität gewährleistet. Das heißt auch implizit, dass es aufgrund dieser Arbeitsweise der Versionsschnittstelle keine dynamische Referenz geben kann, die nicht auf die letzte Version eines Revisionsstranges zeigt. Nur statische Referenzen können damit auf Versionen zeigen, die schon Versionsnachfolger haben.

Aktualisieren Nutzer nun Knoten, auf welche statisch verwiesen wird, werden diese Referenzen nicht aktualisiert. Wünscht jedoch ein Nutzer, dass auf diesem Strang auf die aktuelle Version verwiesen wird, so muss auch von den Knoten eine neue Version erstellt werden, die eine statische Referenz auf den aktualisierten Knoten hatten. Somit sind bei Einstellung eines neuen Seitenstrangs, auf den dynamisch referenziert werden soll, zwei Schritte durchzuführen:

1. Aktualisierung der Version mit Bildung eines Seitenstranges. Dies entspricht dem normalen Aktualisierungsvorgang eines Knotens.

2. Setzen dynamischer Referenzen auf diesen neuen Seitenast, die auf den aktualisierten Knoten verwiesen haben. Hier findet auch eine Aktualisierung statt, da nun Referenzen verändert wurden.

Diese zwei Schritte können von der Versionsschnittstelle in einem Schritt bewerkstelligt werden, wenn der Nutzer angibt, dass er vom neuen Seitenast immer die aktuelle Version haben möchte. Dabei können drei Fälle unterschieden werden:

- Wenn für einen Ressourcenbaum ein neuer Seitenast erstellt wird, so muss je für den Layout-, Ressourcen- und Präsentationsbaum, welcher auf diesen neuen Seitenast verweisen soll, eine neue Version erzeugt werden und die entsprechende Referenz auf den Typ "current" gestellt werden.
- Wenn für einen Layout-Baum ein neuer Seitenast erstellt wird, so muss der Präsentationsbaum aktualisiert werden, welcher eine dynamische Referenz auf diesen neuen Seitenast haben soll.
- Bei Bildung eines neuen Seitenastes in einem Präsentationsbaum sind andere Knoten nicht von einer Aktualisierung betroffen.

In allen drei Fällen muss der Nutzer, der auf eine nun aktualisierte Präsentation dynamisch verwies, benachrichtigt werden, damit er entscheiden kann, ob er die neue Präsentation mit benutzt, die auf einen neuen Seitenast im Layout- oder Ressourcenbaum verweist. Dabei muss dem Nutzer klar mitgeteilt werden, dass nicht eine gewöhnliche Aktualisierung eines Präsentationsinhaltes vorgenommen wurde, sondern dass nunmehr eine bisher statische Referenz auf eine alternative Version des Knotens zeigt. Daher darf bei dieser Aktualisierung die dynamische Referenz eines Nutzers auf die Präsentation nicht aktualisiert werden. Die Aktualisierung muss dem Nutzer überlassen bleiben. Diese Vorgänge veranschaulicht Abbildung 29.

Wenn keine weiteren Versionen durch die Autoren direkt erzeugt werden, bedeutet dies trotzdem für die Anzahl der Präsentationsversionen, dass sie mit der Anzahl der Revisionsstränge der durch sie referenzierten Layouts und Ressourcen wächst. Die Anzahl der Layoutversionen entspricht analog den Präsentationen der Anzahl der Seitenstränge der Ressourcen, auf die es verweist. Bei den Ressourcenknoten ist es ähnlich: hier ist die Anzahl der Versionen auch immer mindestens die Anzahl der Seitenstränge, auf die die jeweilige Ressource verweist.

Dies hängt damit zusammen, dass für Aktualisierungen auf einem Revisionsstrang keine neuen Versionen der Knoten erzeugt werden, die auf diese aktualisierten Knoten dynamisch verweisen. Ansonsten müsste für jede dynamisch referenzierte Version eines Knotens eine neue Version der referenzierenden Knoten erstellt werden.

Diese Form der Handhabung von Seitenästen, hat somit den entscheidenden Vorteil, dass eine Aktualisierung der referenzierenden Knoten nicht immer vorgenommen werden muss, sondern nur dann, wenn ein Seitenstrang erzeugt wird, der referenzierende Knoten eine statische Referenz hatte und der Nutzer nun dynamisch auf diesen Seitenast verweisen will. Currency ist gewährleistet, und gleichzeitig haben Nutzer die Möglichkeit, Seitenäste von Versionen zu erzeugen.

Nach erfolgtem Update aller Knoten und Referenzen müssen noch die Nutzer benachrichtigt werden, die jene Präsentationen benutzen, die auf die aktualisierten Knoten referenzieren.

Auslesen von Inhalten und Metadaten über die Inhalte

Beim Auslesen von Inhalten geht es darum, der Applikation, welche auf der privaten Datenbank arbeitet, die Versionsinformationen zu verbergen. Ressourcen müssen als streng hierar-

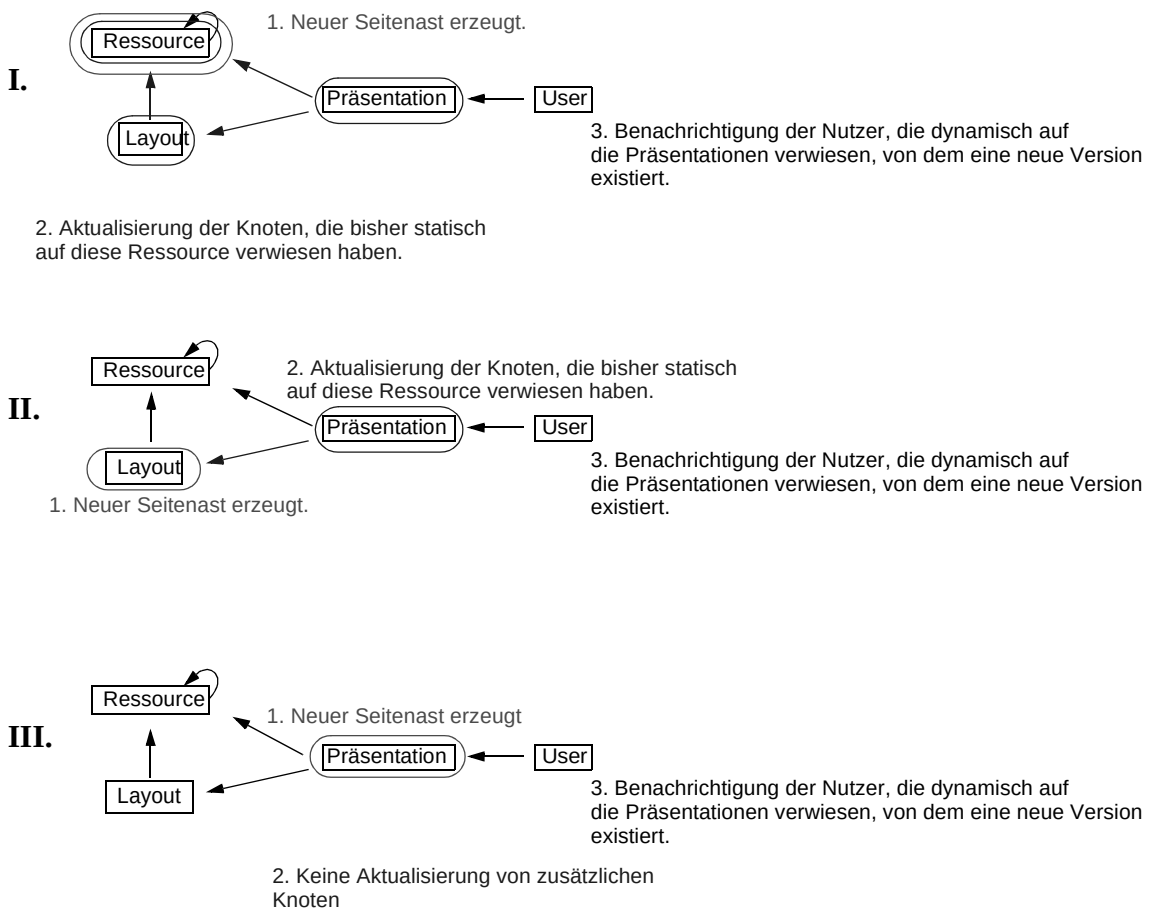

Abbildung 29: Aktualisierung der Referenzen bei Erzeugung von Seitenästen

chisch strukturierte Bäume ohne eine Traversierung durch *pseudo-part-of*-Referenzen erscheinen. Diese Bäume sind dann in der privaten Datenbank in die entsprechende Ressourcenliste einzufügen.

Gleichzeitig dürfen die Versionsknoten mit den Informationen zu Versionsnachfolgern und -vorgängern nicht ausgelesen werden und müssen durch die Versionsschnittstelle ausgelassen werden. Ist von der Applikation erwünscht, die Versionsnachbarn eines bestimmten Knotens zu ermitteln, so schickt es eine Anfrage an die Versionsschnittstelle, die bezeichnet, von welchem Objekt die Versionsnachbarn gesucht sind. Die Aufgabe der Versionsschnittstelle ist dann, ausgehend vom angegebenen Knoten den Versionsbaum auszugeben.

Die Bäume selbst enthalten jedoch auch Referenzen auf andere Knoten und in diesen Referenzen sind Versionsinformationen enthalten. Diese müssen so ausgegeben werden wie sie sind und dürfen weder verändert noch aufgelöst werden. Die Auflösung der Referenzen geschieht somit in einem nächsten Schritt, indem die Knoten in die private Datenbank eingelesen werden, auf die verwiesen wird.

Änderungen der Nutzereinstellungen und Notifikation

Will ein Nutzer seine persönlichen Einstellungen ändern, so wird zunächst eine Anfrage an die Versionsschnittstelle gestellt. Dieser gibt die jetzigen Nutzereinstellungen an die Applikation

zurück, indem es die dem Nutzer entsprechenden Einstellungen aus der öffentlichen Datenbank ausliest und diese Daten in die private Datenbank einfügt. Diese können dann von einer Applikation gelesen und durch den Nutzer bearbeitet werden. Ändert ein Nutzer beispielswiese sein Passwort, so wird das neue Passwort durch die Versionsschnittstelle in die öffentliche Datenbank eingefügt.

Bei Änderungswünschen bezüglich der benutzten Präsentationen stellt sich die Situation ähnlich dar. Möchte ein Nutzer seine Referenz auf eine Präsentation ändern, so wird die vorher eingelesene Referenz im Nutzerprofil auf einen neuen Wert gesetzt. Gleichzeitig wird die Nutzerhistorie im Knoten *user.history* um einen weiteren Listeneintrag *user.context* erweitert. Der aktuelle Kontext des Nutzers im Knoten *user* wird gleichzeitig auf den neuen Inhalt gesetzt. Dadurch ist es möglich, eine Art "Undo"-Funktion zu ermöglichen, die bei Wunsch des Nutzers seine vorher benutzte Präsentation wieder abruft, indem in der *user.history* ein ehemals benutzter *user.context* als neuer Kontext des Nutzers fungiert. Anhand der Durchnummerierung des *user.context* im Attribut *number* ist es der Versionsschnittstelle möglich, die Vorgänger und Nachfolger herauszufinden. Die Versionsschnittstelle muss auch hier die Kapselung der Nummerierung übernehmen und bei jeder neu benutzten Präsentation die Liste der bisher benutzten Präsentationen fortlaufend numerieren. Außerdem darf die Kontext-Liste nicht versehentlich gelöscht werden, daher ist der Zugriff auf diese Listen zu beschränken.

Nach erfolgtem Update aller Knoten muss die Versionsschnittstelle auch die Benachrichtigung aller betroffenen Nutzer durchführen. Bei der Handhabung der Notifikationseinstellungen ist das Benachrichtigungsmodell aus Tabelle 5 zu berücksichtigen. Der Nutzer kann unter *notification* für die Art der Benachrichtigung das Attribut *notification.type* entweder auf "Push" oder "Pull" stellen. Von der Applikationsseite her muss sichergestellt werden, dass ein Nutzer bei der Push-Benachrichtigung immer eine E-Mail bekommt, d. h. durch Kapselung mittels dieser Schnittstelle muss es dem Nutzer verboten sein, eine Push-Benachrichtigung mittels Update-Flags zu erhalten.

Anhand von *notification.type* entscheidet das Versionierungssystem nach einem erfolgten Update, in welcher Form die Benachrichtigung stattfinden soll. Soll eine Pull-Benachrichtigung erfolgen, so werden die Änderungsindizien im Nutzerprofil unter *updates.list* gespeichert. Diese enthalten alle relevanten Daten über die aktualisierten Knoten, wie z. B. *resource.id*, *update.by*, *update.date* etc.

Für die globale Änderungswarnung wird das Attribut *recent.updates* im Knoten *notification* auf "Yes" gesetzt, welches dafür sorgt, dass der Nutzer beim Einloggen eine Warnung bekommt, dass eine Änderung stattgefunden hat. Dieses Attribut wird dann von der Versionsschnittstelle beim Einloggen des Nutzers ausgelesen und interpretiert. Ist dieses Attribut gesetzt, so teilt die Versionsschnittstelle der Applikation mit, dass eine Änderung stattgefunden hat und gleichzeitig wird die Liste aller Updates an die Applikation geschickt, so dass sie die globale Warnlampe leuchten lässt und im Kontrollmodus anhand dieser Liste alle veränderten Knoten markiert. Nach erfolgter Warnung und Bestätigung durch den Nutzer wird dieses Attribut wieder auf "No" zurückgesetzt. Die Update-Liste ist durch die Versionsschnittstelle ebenfalls nach erfolgter Benachrichtigung zu löschen.

Soll hingegen statt der Änderungsindizierung eine E-Mail Nachricht verschickt werden, so wird gleich nach den Updates die E-mail generiert und an den Nutzer verschickt, ohne dass die Nachricht im Nutzerprofil gespeichert wird.

7.4 Prototypische Implementierung

In diesem Kapitel werden exemplarisch Lese- und Schreiboperationen der Versionsschnittstelle implementiert. Eine Ressource wird ausgelesen und nach Aktualisierung wieder in die Datenbank eingefügt. Dabei werden Transformationen der Datenbank durchgeführt. Die Datenbank wird für diese prototypische Implementierung durch eine XML-Struktur realisisert, die für jeden der verschiedenen Schritte jeweils in einer Datei abgelegt wird. Die hierfür benutzte Transformationssprache ist die für XML Datenstrukturtransformationen konzipierte Skriptsprache XSLT.

Die durchgeführten Transformationen lehnen sich dabei an den im vorhergehenden Kapitel spezifizierten Transformationen zur Aktualisierung und zum Lesen von Ressourcen sowie die Aktualisierung von dynamischen Referenzen an.

Die in Kapitel 7.3 gestellten Anforderungen an die Implementierung des Versionskontrollsystems sind im wesentlichen zweigeteilt:

- Anforderungen an die Versionsapplikation: Diese beinhalten den Kontrollmodus sowie die dazu gehörigen Editiermöglichkeiten an den Präsentationen. Die Applikation stellt die Anfragen an die Versionsschnittstelle.
- Anforderungen an die Versionsschnittstelle: Diese beinhaltet die Forderungen an die Lese- und Schreiboperationen der Versionsschnittstelle, welche mit der auf diese zugreifenden Applikation verbunden ist.

Das Versionskontrollsystem muss somit diese beiden Bereiche abdecken. Die detaillierten Forderungen bezüglich der Implementierung der Versionsschnittstelle finden sich in Kapitel 7.3. Aus diesen ist auch ersichtlich, welche Eigenschaften die Applikation haben muss, um dem Nutzer eine Schnittstelle zur Versionierung bereitzustellen. Bei der Ausgestaltung der Applikation kommt es darauf an, diese möglichst intuitiv zu gestalten.

Die Umsetzung der Präsentationserzeugung und implementierungstechnischen Aspekte des Exports von Präsentationen wurden in Kapitel 4 diskutiert. Diese beinhalten ein Document Object Model (DOM), auf das eine Applikation zugreift sowie die Datenbankanbindung dieser. Diese Implementation wird separiert von der Datenbank ohne Versionsinformationen. Die Aufgabe der Implementation eines Versionskontrollsystems ist es hier, sich zwischen der Datenbank mit und ohne Versionsinformationen einzugliedern. Die Ausgestaltung der Schnittstelle könnte jedoch ähnlich sein. Die Benutzung einer plattformunabhängigen Programmiersprache wie JAVA bietet sich beispielsweise an. Ebenso wäre es vorteilhaft aufgrund der Modellierung der Daten in XML diese in ein Document Object Model (DOM) für Applikationen bereitzustellen, so dass diese die Daten hierüber austauschen können.

Die Zielarchitektur für das Versionskontrollsystem ist in Abbildung 30 illustriert.

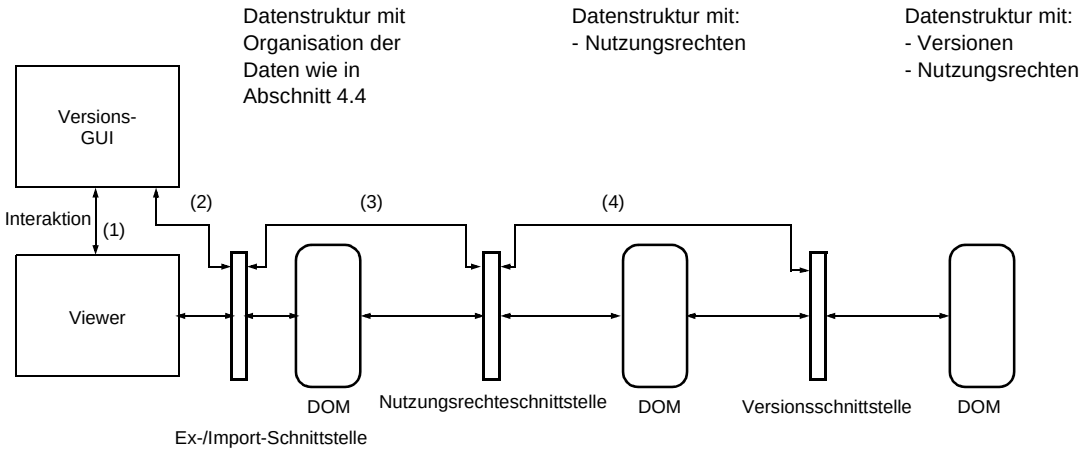

Abbildung 30: Zielarchitektur des Versionskontrollsystems

Wünscht ein Nutzer beispielsweise die Aktualisierung einer Ressource in einer Präsentation, so geht er in den Editiermodus und ruft somit die Graphische Nutzer Schnittstelle (GUI) auf, die mit der benutzten Darstellungsapplikation kommuniziert. Wünscht sich der Nutzer an einer bestimmten Stelle der Präsentation eine Aktualisierung, so muss dieser diese Stelle kenntlich machen (z.B. durch Doppelklick), so dass der Viewer diese der GUI mitteilen kann. Diese kommuniziert mit der Schnittstelle zum Ex- und Import der Präsentationen, um die zu ändernde Ressource zu identifizieren. Diese identifiziert die zu ändernde Ressource und fragt bei der nächsten Schnittstelle nach, ob Schreibrechte auf diese Ressource für den Nutzer vorhanden sind. Ist dies nicht der Fall, so wird an die GUI eine Meldung zurückgegeben, dass ein Verändern der Ressource aufgrund der Rechte des Nutzers nicht möglich ist. Ist dies jedoch möglich, so übernimmt die Nutzungsrechteschnittstelle die vorgenommen Änderungen von der Ex- und Importschnittstelle und gibt diesen Änderungsbefehl mit der Angabe des Nutzers, der zu ändernden Ressource, den Zeitangaben und den geänderten Inhalten an die Versionsschnittstelle weiter. Diese fügt dann die entsprechenden Änderungen in die Versionsstruktur ein, aktualisiert alle zugehörigen Referenzen und sorgt für die Benachrichtigung der Nutzer, die von dieser Änderung betroffen sind. Diese einzelnen Schritte sind in Abbildung 30 chronologisch numeriert dargestellt.

Die Angaben, die die Versionsschnittstelle für das Schreiben oder Lesen aus der öffentlichen Datenbank benötigt, werden von der Nutzungsrechteschnittstelle geliefert und liegen somit gekapselt vor. Die Versionsschnittstelle kann sich auf die Aktualisierungs-, Lese-, und Notifikationsaufgaben beschränken, was die Implementierung dieser Schnittstelle vereinfacht.

Geht man davon aus, dass die benötigten Operationsparameter für die Versionsschnittstelle vorliegen, so kann eine Implementation prototypisch erfolgen. Im Rahmen dieser Arbeit wurden daher zu Demonstrationszwecken die grundlegenden Transformationen der Datenstruktur genauer spezifiziert. Hier wird mittels XML-Transformationen gezeigt, wie die Änderungen bei Lese- und Schreiboperationen aussehen werden. Dabei beschränkt sich die Spezifikation

auf Aktualisierungs- und Leseoperationen von *resource*-Elementen, die nur andere *resource*- und *blob*-Elemente enthalten.

7.4.1 Durchgeführte XML-Transformationen und ihre Spezifikation

Beispielhaft wurde in Anlehnung an Abbildung 27 das Auslesen und Aktualisieren einer Ressource mittels Datenstrukturtransformationen umgesetzt. Die Transformationen der privaten und öffentlichen Datenbanken wurden mit der Extensible Layout Sheet Language (XSL) durchgeführt, da die Datenbankinhalte als XML-Textdateien vorlagen. Die umzusetzenden Befehle sind "insertnew", "change", "delete" und "retrieve", die Inhalte in der öffentlichen Datenbank verändern sollen oder aus ihr in die private Datenbank auslesen sollen.

Im Anhang werden die Elemente der durchgeführten prototypischen Implementierung im Einzelnen präsentiert und erläutert.

7.5 Zusammenfassung

Ziel dieses Kapitels war es, ein Versionsmanagementsystem für das Szenario des verteilten Vorlesungsarchivs in Abschnitt 2.1.3 zu konzipieren. Dieses Ziel war aus den drei Hauptanforderungen abgeleitet. Diese sollen im folgenden nochmals erwähnt werden und deren Zielerreichung durch das hier erarbeitete Konzept belegt werden.

Historie der Inhalte: *Dem Nutzer muss es ermöglicht werden, unter mehreren Varianten individuell wählen zu können, d.h. eine neue Präsentation darf eine alte Präsentation nicht unwiderruflich löschen, sondern muss diese ergänzen.*
Im Rahmen dieser Arbeit wurden die bereits vorhandenen Ressourcen- , Layout- und Präsentationsbäume um Versionsinformationen erweitert. Diese dienen dazu, die im Laufe der Zeit voneinander abgeleiteten Inhalte in eine Reihenfolge zu bringen, so dass die Entstehungsgeschichte für jede Version eines Knotens genau verfolgt werden kann. Durch Kapselung der Änderungsoperationen mittels einer Versionsschnittstelle ist dafür gesorgt, dass bei ausschließlicher Benutzung dieser Schnittstelle die Aktualisierung oder Veränderung von Ressourcen nicht zu einem Verlust der Inhalte, sondern zu einer Erweiterung der insgesamt gespeicherten Informationen führt. Ebenso gewährleistet die Schnittstelle beim Auslesen der Inhalte die Auflösung von vorhandenen Referenzen und das Verbergen der Versionsinformationen.

Konsistenz der Ressourcen: *Die Aktualisierung von Inhalten wird durch die Nutzer unabhängig voneinander vorgenommen. Es müssen Mechanismen vorhanden sein, die ein Durcheinander der Aktualisierungen verhindern. Nutzer müssen von den Änderungen erfahren.*
Die Versionsschnittstelle gewährleistet die Konsistenz durch Benachrichtigung der Nutzer und Aktualisierungspropagation bei den sogenannten entfernten Updates. Durch Setzen von Zugriffsstati für einzelne Versionen können gleichzeitige Zugriffe auf einen Knoten vermieden werden. Durch ein von den Nutzern konfigurierbares Benachrichtigungssystem erfahren die Nutzer von Änderungen, die Sie betreffen oder von denen Sie eine Benachrichtigung wünschen. Darüberhinaus werden dynamische Referenzen auf Knoten vom Versionierungssystem immer auf dem aktuellen Stand gehalten.

Nutzer-Historie: *Jeder Nutzer sollte seine Präsentationen individuell verfolgen können.*

Falls der Wunsch besteht, vorher benutzte Inhalte wiederzuerlangen, so sollte dies einfach möglich sein.
Die individuellen Nutzerprofile werden in einer Nutzerliste gespeichert, worin alle nutzerspezifischen Daten wie z.B.: User-ID, Passwort, Nutzerdaten etc. enthalten sind. Hier wurde auch eine Liste eingefügt, die alle bisher benutzten Präsentationen des Nutzers enthält. Damit ist es der Versionsschnittstelle möglich, bei entsprechendem Befehl des Nutzers die aktuell benutzte Präsentation auf eine bisher benutzte oder eine neue Präsentation zu setzen. Indem diese Liste des Nutzer nie gekürzt sondern nur erweitert wird, ist ein Zurücksetzen auf jede beliebige, bisher benutzte Präsentation möglich.

Ausgangspunkt der Arbeit war die Organisation der Daten in Abschnitt 4.3. Die Daten sind mittels der standardisierten Dokumentenbeschreibungssprache XML modelliert, sind somit streng hierarchisch organisiert. Doch es gibt auch Querverweise der Inhalte untereinander. Hier haben sich aufgrund der Szenarienbetrachtung im zweiten Kapitel vier Bereiche, die für eine Versionierung interessant sind, herauskristallisiert: Ressourcen-, Präsentations-, Layoutlisten sowie die Referenzen. Durch Betrachtung der Szenarien hat sich ergeben, dass eine Dynamisierung der Referenzen und die Versionierung der Listen notwendig ist. Desweiteren hat sich die Notwendigkeit der Koordination der kollaborativen Tätigkeiten der Nutzer untereinander ergeben, da bei unabhängiger Nutzung der Präsentationen und deren Aktualisierung Probleme bei der Konsistenz der Präsentationen zu befürchten sind. Der Wunsch der Nutzer nach permanenter Aktualität der Präsentationen bei gleichzeitiger Bewahrung vor unerwarteten Inhalten bildete den Ausgangspunkt für die Überlegungen zum Benachrichtigungsmodell, welches ebenfalls im zweiten Kapitel entwickelt wurde.

Somit wurde im zweiten Kapitel herausgearbeitet, welche Bereiche der Datenstruktur verändert werden müssen, um die geforderten Funktionalitäten aufgrund der Nutzungsszenarien realisieren zu können.

Da eine Vielzahl von Systemen zur Konsistenzerhaltung von Daten und Koordination von kollaborativen Tätigkeiten bestehen, wurden im dritten Kapitel Software Configuration Management (CM) Systeme sowie die Versionierungsmodelle von Datenbankmanagementsystemen (DBMS) betrachtet, um eventuell aus diesen Methoden bewährte Modelle übernehmen zu können. Die Betrachtung dieser zwei Bereiche hängt damit zusammen, dass besonders an großen Softwareprojekten viele Entwickler beteiligt sind, und Softwareentwicklung eine Vielzahl von Teilergebnissen innerhalb der Entwicklung benötigt, damit ein fertiges Produkt entwickelt werden kann. Außerdem müssen beispielsweise verschiedene Ausprägungen von derselben Software vorhanden sein, um auf verschiedenen Plattformen einsetzbar zu sein. Somit gehören aufgrund der Komplexität von Softwareentwicklungsprojekten die CM Systeme zu den leistungsstärksten Instrumenten zur Koordination von Arbeitsabläufen in der Entwicklung von Designartefakten.

Hier hat sich jedoch gezeigt, dass eine Vielzahl der Koordinationsinstrumente aufgrund der relativ einfachen Anforderungen aus dem zweiten Kapitel nicht benötigt werden. Ein einfaches Änderungsmanagement mit Versionskontrolle der Präsentationen und deren Inhalten genügte hier. Weder eine Vorstrukturierung von Arbeitsabläufen entlang von Entwicklungslinien von Designartefakten, noch die Generierung von Statusreporten etc. waren für eine Umgebung wie in Abschnitt 2.1.3 notwendig.

Versionskontrolle in CM Systemen hat zwar eine grundlegende Bedeutung, jedoch ist ein Großteil der Funktionalitäten auf die Koordination der Arbeiten konzentriert. Anders hingegen stellt sich die Situation im Falle der DBMS dar, da diese auf Ebene der Inhaltsobjekte durch Versionsmodelle die Konsistenz der Daten, sowie der Struktur der Daten sicherstellen. Dies ist insbesondere in CAD Datenbanken notwendig, wo eine starke Verknüpfungsdichte der Designartefakte untereinander auf der einen Seite und eventuell häufige Aktualisierungen von Artefakten auf der anderen Seite vorherrschen. Hier wird Konsistenz der Daten durch Propagation der Aktualisierungen entlang der Verknüpfungen erreicht. Es stellt sich auch das Problem beim Vorhandensein von Seitenästen in Versionsbäumen, welche Version nun für den Nutzer die Aktuelle ist. Diese war ebenfalls hier zu definieren. Bei der Zusammenstellung von dynamischen Konfigurationen, welche für die Erstellung multimedialer Inhalte relevant sind, gab es ebenfalls vielversprechende Konzepte, welche für eine Adaptation in Erwägung gezogen wurden.

Nachdem alle relevanten Bereiche aus den beiden Systemen, in deren Versionskontrolle die Grundlage der Anwendung bildet, diskutiert wurden, wurde in einem letzten Abschnitt die Relevanz und Anwendbarkeit der Modelle evaluiert. Hier ergab sich dann ein Gesamtergebnis des vierten Kapitels, welches in Tabelle 12 zusammengefasst ist. Ab dann konnte eine Adaption der Ergebnisse Kapitel 5 erfolgen.

Die Spezifikation des Zielsystems der Versionierung beinhaltete zwei Bereiche: Erweiterungen der bisherigen Datenstruktur unter Berücksichtigung der gewünschten Funktionalitäten aus dem zweiten Kapitel sowie die Spezifizierung der Versionsschnittstelle, die einerseits die Konsistenz der Daten sicherstellen und auf der anderen Seite die Operationen der Versionskontrolle kapseln soll. Besondere Berücksichtigung fand dabei eine Besonderheit der Datenstruktur aus Abschnitt 4.3: Präsentationen verweisen auf Ressourcen und Formatvorlagen, diese verweisen wiederum auf Ressourcen, diese können wiederum auf andere Ressourcen verweisen. Die Nutzer verweisen auf Präsentationen. Da diese Referenzen durchweg dynamisiert wurden, stellte sich die Frage, wie einerseits aus dieser Struktur die Konsistenz der Präsentationen bei Aktualisierungen gewährleistet wird und andererseits welche Version der Präsentation für den Nutzer als aktuelle gilt. Dies wurde in Kapitel 7.3.4 festgesetzt. Hiernach ist die aktuelle Version für die meisten Nutzer die letzte Version auf dem Hauptrevisionsstrang nach Kapitel 28. Da Seitenäste im Versionsbaum vorhanden sind, können einige Nutzer auch auf Nebenrevisionssträngen die letzte Version als aktuelle Version benutzen. Diese Unterscheidung hängt damit zusammen, dass davon auszugehen ist, dass die meisten Nutzer von vorne herein die aktuelle Version einer Ressource benutzen und aktualisieren werden und sich somit dieser Hauptrevisionsstrang bildet, aber Seitenäste nicht auszuschließen sind. Hervorzuheben ist hierbei, dass bei Aktualisierung einer dynamischen Referenz auf eine neue Version in den wenigsten Fällen die Erzeugung einer neuen Version des Baumes nach sich zieht, in welchem diese Referenz auftaucht. Bei Aktualisierungen ist jedoch der Präsentationsinhalte bei Nutzern von Präsentationen mit dynamischen Referenzen immer eine Benachrichtigung dieser vorzunehmen, um den Nutzern unerwartete Inhalte zu ersparen.

Eine prototypische Implementation der Schreib- und Leseoperationen von Ressourcen des präsentierten Versionierungssystems zeigte die Geschlossenheit des Konzepts. Dabei ging es

für den einfachsten Fall um das Auslesen eines Ressourcenbaumes, die Aktualisierung dieser durch den Nutzer sowie das Einspeichern des Baumes in die öffentliche Datenbank mit Aktualisierung der dynamischen Referenzen. Diese Implementation betraf somit die grundlegenden Transformationen der Daten in der Datenbank und wurde mit der Datentransformationssprache XSLT (Extensible Layout Sheet Language Transformations) codiert.

Kapitel 8 - Zusammenfassung und Ausblick

Für große, verteilte multimediale Anwendungen müssen oft die Ergebnisse vieler Autoren erzeugt, gepflegt und integriert werden. Um aus dem hohen Aufwand für diese Beiträge den Inhalt eines Gesamtsystems erhalten zu können, müssen diese Beiträge formalen und vor allem auch inhaltlichen Kriterien genügen, etwa einer Corporate Identity und inhaltlicher Konsistenz.

Diese Arbeit zeigte anhand konkreter Beispiel-Applikationen, dass die vorhandenen Techniken etwa zur Archivierung von Daten, zur Synchronisation von Teams und zur Integration von Inhalten hier anwendbar sind, aber erheblich angepasst bzw. neu kombiniert werden müssen. Vor allem die reiche Struktur multimedialer Inhalte erfordert Anpassungen, sie ist gekennzeichnet durch viele relevante technische und inhaltliche Attribute solcher Medien (Schriftgröße, Sprache, Format, benötigte Bandbreite) und die vielfältigen Beziehungen in einem multimedialen Dokument (enthalten-sein, referenzieren, auch zeitliche Beziehungen in Präsentationen). Multimediale Autoren bearbeiten oft spezialisiert nicht nur verschiedene Inhalte, sondern auch klar unterscheidbare Aspekte desselben Inhalts. Designer werden Farben und Größen von Präsentationselementen ändern, Journalisten eher die Inhalte.

Die Arbeit untersuchte in den ersten Kapiteln auftretende Anforderungen und wie weit vorhandene Techniken adäquat sind, Lösungen für komplette Systeme oder für Teilaspekte zu realisieren. Das Ergebnis war, dass innerhalb der vielen relevanten Themenfelder ein Schwerpunkt an offenen Problemen auf der Applikationsebene liegt, in den Bereichen Rechtemanagement, Benutzerrollen und Inhaltsversionierung.

Als Basis zur Realisierung und Integration entsprechender Lösungen wurde dann eine explizite Darstellung der multimedialen Daten, der Information über die Autoren und des Systemzustandes erarbeitet. Eine entsprechende Datenstruktur wurde zusammen mit einer Architektur entwickelt, in der diese Struktur von der physikalischen Darstellung abstrahierbar verwendet wird.

Rechtesysteme sind hier notwendig, müssen allerdings die inhaltliche Struktur und auch die besonderen auftretenden Nutzungsrechte bei multimedialen Inhalten berücksichtigen. Es wurde eine Analyse der Anwendbarkeit existierender Rechtekonzepte und ein eigenes Konzept erarbeitet, das die spezifischen Bedürfnisse der Beispielanwendung eines verteilten Archivs für multimediales Vorlesungsmaterial beantwortet.

Weiterhin zeigte die Arbeit, dass die reiche Struktur multimedialen Inhalts zur Definition von Benutzerrollen genutzt werden kann, um verschiedene Arbeitsvorgänge zu unterstützen. Über existierende Systeme hinausgehend präsentierte die Arbeit hier ein Konzept, das zum einen eine explizite, editierbare Repräsentation solcher Rollen bietet, die orthogonal zu Zugriffsrechten die Möglichkeiten einer Änderungssitzung auf die Anforderungen des jeweiligen Arbeitsschrittes fokussieren, also etwa auf inhaltliche, administrative oder Layouttätigkei-

ten. Anders als bei GroupWare- und CSCW-Systemen wurden hier nicht die dynamischen Workflow-Abfolgen, sondern statische Eigenschaften der Inhalte betrachtet.

In einem weiteren Schritt wurde die Versionierung der multimedialen Inhalte eines großen verteilten Projekts auf spezielle Anforderungen untersucht. Das dazu im Rahmen dieser Arbeit realisierte Konzept kennt weniger semantische Vorgaben für Korrektheit als etwa Systeme zur kooperativem Entwicklung von Programmtexten, und es bietet nur eine vereinfachte Propagierung von Änderungen gegenüber allgemeinen Datenbanksystemen. Dies erlaubt es dann aber, den Autoren eine (konfigurierbar) vereinfachende Versionssemantik anzubieten, die den Bedürfnissen der betrachteten Beispielapplikation entspricht.

Als generelles Ergebnis dieser Arbeit wurde der Nutzen einer Abstraktionsschicht gezeigt, die oberhalb der vorhandenen Speicher-, Verteilungs-, Transport- und Darstellungstechniken speziell die multimedialen und anwendungsspezifischen Eigenschaften einer Applikation modelliert, um in die Technik verteilter multimedialer Systeme Konzepte für höhere Applikationsschichten zu integrieren, in diesem Falle für Rechte, Rollen und Versionen.

Ausblickend auf weitergehende Arbeiten sollte zunächst der Nutzen der vorgestellten Ergebnisse im produktiven Einsatz evaluiert werden.

Dann sollten auch Aspekte der niedrigeren Ebenen einer verteilten multimedialen Anwendung in Einklang gebracht werden, aufbauend auf der hier erarbeiteten Zielvorgabe für Rechte, Rollen und Versionen innerhalb einer Applikation. Fehlerhafte, nicht-ideale Verteilung von Daten wurde in dieser Arbeit nicht betrachtet, die vorgestellten Mechanismen setzen alle auf vollständiger, pessimistischer Verteilung auf, insbesondere wird angenommen, dass ein Inhaltselement effektiv zur Bearbeitung gegen konkurrierende Zugriffe gesperrt werden kann. Hier sollte untersucht werden, welchen Einfluss die notwendigen konfliktauflösenden oder konfliktvermeidenden Strategien auf Netzwerk- und Speichersystemebene zur Propagierung dieser Sperren eventuell auf die vorgestellten Mechanismen haben.

Diese Arbeit lieferte einen Entwurf, der die Darstellung von Inhalts-, System- und Benutzerdaten integrierte, um diese zueinander in Beziehung setzen zu können. Auf diese Art kann zum Beispiel die Generierung von Präsentationen an das Benutzerprofil und an den Systemzustand angepasst werden, wie das Projekt *medianode* demonstrierte [34]. Ein weiterer wichtiger Beleg für die Tragfähigkeit des Ansatzes wäre eine vollständige Integration der Signalisierung auch kurzlebiger Applikationsdaten über die vorgeschlagenen Schnittstellen und Datenstrukturen. Ein erster Ansatz dazu wurde in den Arbeiten von On et.al. [10] und [39] präsentiert.

Möglicherweise liegt ein großer Gewinn darin, die in den vorgestellten Mechanismen explizit vorliegenden Informationen (etwa über Zugriffsmöglichkeiten der Nutzer und Autoren, der semantischen Gleichheit alternativer Medien, Ähnlichkeiten zwischen Versionen u. ä.) an Mechanismen tieferer Systemebenen zu signalisieren. Es wäre zu untersuchen, wie stark zum Beispiel die Effizienz von Replikationsmechanismen oder von Datensicherung durch solche Informationen gesteigert werden kann.

Anhang A - Referenzen

[1] S. Abel. Rechtemanagement für den verteilten Medienserver Medianode. Studienarbeit, Fachgebiet Industrielle Prozeß- und Systemkommunikation, Technische Universität Darmstadt, Februar 2000, KOM- S-0065

[2] G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language Reference Manual. Object Technology Series, Addison Wesley, USA 1998

[3] D. Box. Essential COM. Menlo Park, CA: Addison-Wesley Publishing Company, 1998. ISBN 0-201-63446-5.

[4] S. Dart. Spectrum of Functionality in Configuration Management Systems. Technical Report, Software Engineering Institute, Carnegie Mellon University Pittsburgh USA, 1990

[5] R. Eberhardt: A Multimedia Device and Stream Management Architecture Based on Distributed-Object Computing Technology. Dissertation an der Universität Darmstadt, November 2000.

[6] ECMA-262, ECMAScript. European Computer Manfacturers Association, Juni 1998.

[7] J. Estublier, R. Casallas. Three dimensional versioning. In Estublier, Jacky (Ed.): Software Configuration Management: selected papers / ICSME SCM-4 and SCM-5 workshops. Volume 1005 of Lecture Notes in Computer Science, Seattle, Washington, October 1995. Springer-Verlag, S. 118-135

[8] A. Ford. mod_perl Pocket Reference,OReilly & Associates Inc., New York, 2001

[9] C. Griwodz, M. Liepert, and The Hynode Consortium. Personalised News on Demand: The HyNoDe Server. In Proc. of the 4th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS'97), volume 1309, pages 241-250. Springer-Verlag, Berlin, Heidelberg, New York, September 1997.

[10] C. Griwodz, O. Merkel, J. Dittmann, R. Steinmetz. Protecting VoD the Easier Way. In Proc. of ACM Multimedia 1998, pages 21-28, September 1998.

[11] C. Griwodz, M. Zink, M. Liepert, G. On, and R. Steinmetz. Multicast for Savings in Cache-based Video Distribution. In Multimedia Computing and Networking 2000. SPIE, January 2000.

[12] C. Griwodz, M. Zink, M. Liepert, and G. On. Analytical Model of Patching VoD Cache Hierarchies. Technical Report 03, KOM TU Darmstadt, September 1999.

[13] C. Griwodz, M. Zink, M. Liepert, and R. Steinmetz. Position Paper: Internet VoD Cache Server Design. In ACM Multimedia 99 (Part 2), pages 123-126. ACM Press, New York, USA, October 1999. ISBN 1-58113-239-5.

[14] C. Griwodz, M. Liepert, G. On, M. Zink. A Distributed Media Server for the Support of Multimedia Teaching, Industrial Process and System Communications (KOM), TU-Darmstadt, Darmstadt, 1999

[15] C. Griwodz, M. Liepert, M. Zink, and R. Steinmetz. Tune to Lambda Patching. ACM Performance Evaluation Review, 27(4):20-26, March 2000.

[16] E. R. Harold. Java Network Programming. Second Edition. O'Reilly, Sebastopol, 2000

[17] A. Haake, J. M. Haake. Take {CoVer}: Exploiting Version Support in Cooperative Systems. In Proceedings of INTERCHI'93, ACM Press, Amsterdam, The Netherlands, April 24-29. Addison Wesley, 1993. Seiten 406-413

[18] J. Heidemann. Performance Interactions Between P-HTTP and TCP Implementations, in ACM Computer Communication Review, April 1997

[19] W. v. Humboldt. Über das Entstehen der grammatischen Formen und ihren EinfluÛs auf die Ideenentwicklung. Werke Band 3: Über die Sprache. Wissenschaftliche Buchgesellschaft, Darmstadt, 1963.

[20] IBM. *HyNoDe - Implementation of the Multimedia Server,* Status Report, 1997

[21] ISO/IEC IS 10918. Information Technology - Digital Compression and Coding of Continuous-Tone Still Images. ISO/IEC JTC1/SC29, 1993

[22] ISO/IEC IS 13818. Information Technology - Generic Coding of Moving Pictures and Associated Audio Information (MPEG2). ISO/IEC JTC1/SC29, 1996

[23] ISO/IEC CD 14496-6. Information Technology - Generic Coding of Moving Pictures and Associated Audio Information (MPEG4). ISO/IEC JTC1/SC 29 WG 11 N2206, 1998

[24] ISO/IEC 9075. Information Technology - Database Languages - SQL. ISO/IEC, 1992

[25] R. H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. ACM Computing Services, Vol. 22, No. 4, Dezember 1990, S. 375-406

[26] K.-F. Jea, H.-B. Feng, Y.-R. Yau, S.-K. Chen, J.-C. Dai. A Difference-based Version Model for OODBMS. In Proceedings of the Asia Pacific Software Engineering Conference, 1998

[27] J.B.D. Joshi, W. G. Aref, A. Ghafoor, E. H. Spafford: Security Models For Web-Based Applications, Communications of the ACM vol. 44, number 2, pp. 38-44, ACM, New York, February 2001

[28] R. John. Konzeption und Realisierung von Rechtestrukturen für Directory-Systeme im Universitätsumfeld. Studienarbeit, Fachgebiet Industrielle Prozeß- und Systemkommunikation, Technische Universität Darmstadt, Oktober 2001, KOM - S - 0106

[29] J. Kahl. Modellierung eines verteilten Multimediaservers. Fachgebiet Industrielle Prozeß- und Systemkommunikation, Technische Universität Darmstadt, April 2001, KOM - D - 0101

[30] Th. Käppner. Entwicklung verteilter Multimedia-Applikationen : Strategie und Realisierung einer geeigneten Systemunterstützung. Vieweg, 1997. ISBN 3-528-05549-9

[31] Y. Karakaya. Konzipierung eines Versionskontrollsystems für den verteilten Multimediaserver Medianode. Fachgebiet Industrielle Prozeß- und Systemkommunikation, Technische Universität Darmstadt, April 2001, KOM - S - 0067

[32] A. Koumpis, C. Karagiannidis, C. Stephanidis, "Adaptations of the Text Media Type: Addressing the User's Perspective", 18th International Conference on Software Engineering, IEEE International Workshop on Multimedia Software Development, Berlin, Germany, March 25-26, 1996.

[33] T. Kunkelmann. Sicherheit für Videodaten. Disserationsschrift (PhD thesis), Vieweg-Verlag, ISBN 3-528-05680-0, 1998

[34] M. Liepert, C. Griwodz, G. On, M. Zink, and R. Steinmetz. A distributed media server for the support of multimedia teaching. In Multimedia Systems and Applications II, SPIE 99, Boston. Poster presentation, August 1999.

[35] B. Ludäscher, Y. Papakonstantinou, P. Velikhov, V. Vianu. View Definition and DTD Inference for XML. Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats (SSD99), Jerusalem, 1999

[36] Sun Microsystems Inc. JavaTM 2 Platform, Standard Edition, v 1.3.1 API Specification. http://java.sun.com/j2se/1.3/docs/api/, Sun Microsystems Inc., Mai 2001

[37] A. M. Noronha. Extending a Structured Document Model with Version Control. In Proceedings of the International Database Engineering and Applications Symposium, 1998

[38] G. On, M. Zink, M. Liepert, C. Griwodz, J. Schmitt, and R. Steinmetz. Replication for a Distributed Multimedia System. In Proceedings of 8th International Conf. on Parallel and Distributed Systems (ICPADS'01), Juni 2001.

[39] G. On, J. Schmitt, M. Liepert, and R. Steinmetz. Replication with QoS support for a Distributed Multimedia System. In the 27th EUROMICRO Conference, Workshop on Multimedia and Telecommunication (EMMTC'01), September 2001.

[40] C. Reichenberger. VOODOO - A Tool for Orthogonal Version Management. In Estublier, Jacky (Ed.): Software Configuration Management: selected papers / ICSME SCM-4 and SCM-5 workshops. Volume 1005 of Lecture Notes in Computer Science, Seattle, Washington, October 1995. Springer-Verlag, S. 61-79

[41] P. Reuter. Organisation der Daten im Medianode System, Studienarbeit, Fachgebiet Industrielle Prozeß- und Systemkommunikation, Technische Universität Darmstadt, Mai 2000, KOM - S - 0066

[42] K. Rothermel, I. Barth, T. Helbig: CINEMA : an architecture for configurable distributed multimedia applications, Technical Report TR-1994-03, Universität Stuttgart, Abteilung Verteilte Systeme, April 1994

[43] K, Rothermel, G. Dermler und W. Fiederer: Qo{S} Negotiation and Resource Reservation for Distributed Multimedia Applications, Technical Report TR-1996-09, Universität Stuttgart, Abteilung Verteilte Systeme, April 1996

[44] RRZN: UNIX Eine Einführung, 9. Auflage, Regionales Rechenzentrum für Niedersachsen, Universität Hannover, 1994

[45] A. Sayegh. CORBA - Standard, Spezifikation, Entwicklung. O'Reilly Essentials, 1997

[46] C. Seeberg, A. El Saddik, K. Reichenberger, A. Steinacker, S. Fischer, and R. Steinmetz. iTeach - Interactive Teaching and Learning. Appendix to the ACM Multimedia '98 Proceedings, September 1998. ISBN 1-58113-036-8

[47] C. Seeberg, I. Rimac, S. Hoermann, A. Faatz, A. Steinacker, A. El Saddik, and R. Steinmetz. MediBook: Realisierung eines generischen Ansatzes für ein interentbasiertes Multimedia-Lernsystem am Beispiel Medizin. In Tagungsband: Treffen der GI-Fachgruppe 1.1.3 Maschinelles Lernen (GMD Report 114), pages 96-105, September 2000.

[48] C. Seeberg: Modulare Wissensbasen zur Erzeugung adaptiver und kohärenter Lehrdokumente. Dissertation an der Universität Darmstadt, März 2001.

[49] R. Steinmetz: Multimedia-Technologie: Grundlagen, Komponenten und Systeme. Springer Verlag, 3. Auflage, Oktober 2001

[50] R. Steinmetz, C. Seeberg, A. Steinacker. Coherence in the Learning System k-Med. In Proceedings der GLDV-Früjahrstagung 2001, pages 17-27, March 2001.

[51] R. Steinmetz and D. Köhler. Ein verteiltes Multimedia-Kiosksystem: Anforderungen, Architektur und Erfahrungen. Themenheft it+ti, Herbst 1995. eingeladener Beitrag.

[52] A. S. Tanenbaum. Computer Networks, 3rd Edition. Prentice Hall PTR; März 1996. ISBN: 0133499456

[53] W. Tichy.. RCS - A System for Version Control. Software---Practice & Experience, 15(7):637--654, July 1985

[54] Transarc Corporation: AFS User's Guide, Transarc Corporation, Pittsburgh, 1992

[55] Tsichritzis, D. C. and Lochovsky, F. H. (1982).Data Models. Prentice-Hall, Eaglewood Cliffs, NJ.

[56] G.Wahl. UML kompakt. in : Objekt Spektrum 2/1998

[57] W. Wang, J. M. Haake. Flexible Coordination with Cooperative Hypertext. Proceedings of ACM Hypertext'98, 1998, Seiten 245-255

[58] B. Westfechtel. Revisions- und Konsistenzkontrolle in einer integrierten Softwareentwicklungsumgebung, In W. Brauer (Hrsg.): Informatik-Fachberichte Volume 280, Heidelberg, Springer-Verlag, 1991

[59] D. Whitgift. Methods and Tools for Software Configuration Management. John Wiley & Sons, Chichester, UK, 1991

[60] W. Wieczerzycki. Versioning Technique for Collaborative Writing Tools. In: Proceedings of theSeventh International Workshop on Database and Expert Systems Applications, 1996, S. 463 -468

[61] R. K. Wong, H. L. Chau, F. H. Lochovsky. A Data Model and Semantics of Objects with Dynamic Roles. ICDE, 1997. Seiten 402-411

[62] M. Zink, A. Jonas, C. Griwodz, R. Steinmetz. LC-RTP (Loss Collection RTP): Reliability for Video Caching in the Internet. In Proc. of 7th Int'l Conf on Parallel and Distributed Systems (ICPADS): Workshops, pages 281-286. IEEE, Piscatay Way, NJ, USA, July 2000. ISBN 0-7695-0571-6.

WWW-Referenzen

[63] M. F. Ali: HTTP and related protocol,
http://ei.cs.vt.edu/~wwwbtb/book/chap16/httpng.html, 1996

[64] A. Baird-Smith, H. Frystyk, J. Gettys, H. W. Lie, C. Lilley, E. Prud'hommeaux.
Network Performance Effects of HTTP/1.1, CSS1, and PNG,
http://www.w3.org/Protocols/HTTP/Performance/Pipeline.html, 1997

[65] K. Balachander, J. Mogul, D. M. Kristol. Key Differences between HTTP/1.0 and HTTP/1.1,
http://akpublic.research.att.com/library/trs/TRs/98/9se8.39/98.39.1.body.ps, 1998

[66] T. Berners-Lee: Information Management: A Proposal,
http://www.w3.org/History/1989/proposal.html, 1989

[67] T. Berners-Lee: HyperText Transfer Protocol Design Issues,
http://www.w3.org/Protocols/DesignIssues.html, 1991

[68] T. Berners-Lee: The Original HTTP as defines in 1991,
http://www.w3.org/Protocols/HTTP/AsImplemented.html, 1991

[69] T. Berners-Lee: Why a new protocol?.
http://www.w3.org/Protocols/WhyHTTP.html, 1991

[70] T. Berners-Lee: Basic HTTP as defined in 1992,
http://www.w3.org/Protocols/HTTP/HTTP2.html, 1992

[71] T. Berners-Lee: The Worl Wide Web: Past Present and Future,
http://www.w3.org/People/Berners-Lee/1996/ppf.html, 1996

[72] T. Berners-Lee, R. Fielding, H. Frystyk. Hypertext Transfer Protocol -- HTTP/1.0,
ftp://ftp.isi.edu/in-notes/rfc1945.txt, RFC 1945, IETF, 1996

[73] T. Berners-Lee, R. Fielding, H. Frystyk, J. Gettys, J. Mogul. Hypertext Transfer Protocol -- HTTP/1.1.
ftp://ftp.isi.edu/in-notes/rfc2068.txt, RFC 2068, IETF, 1997

[74] T. Berners-Lee, R. Fielding, H. Frystyk, J. Gettys, P. Leach, P. Masinter, J. Mogul.
Hypertext Transfer Protocol -- HTTP/1.1.
ftp://ftp.isi.edu/in-notes/rfc2616.txt, RFC 2616, IETF, 1999

[75] N. Borenstein, N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies,
ftp://ftp.isi.edu/in-notes/rfc1521.txt, RFC 1521, IETF 1993

[76] R. Cailliau. A short History of the Web,
http://www.inria.fr/Actualites/Cailliau-fra.html, 1995

[77] C. Griwodz, M. Liepert, G. On, M. Zink. A Distributed Media Server for the Support of Multimedia Teaching, Fachgebiet Industrielle Prozeß- und Systemkommunikation,Technische Universität Darmstadt, September 1999
URL:http://www.httc.de/medianode/english/medianode.html

[78] K. Claffy, G. Miller, K. Thompson. The nature of the beast: recent traffic measurements from an Internet backbone.
http://www.caida.org/Papers/Inet98/index.html, 1998

[79] R.Dampmann, M.Rupp. *Einführung in XML und RDF.*
http://kreuz.cs.uni-dortmund.de:3410/seminar/rdfxml/index.html, 2000

[80] Datachannel Press Release: XML Takes Religion Out of web Computing by Making Java & Windows Interoperable.
http://www.datachannel.com/news/pr_webbroker.html, 1998

[81] Datachannel Press Release: W3C Formally Acknowledges DataChannel's Submission of WebBroker Specification for Application Linking.
http://www.datachannel.com/news/pr_webbroker_wc.html, 1998

[82] G.Dermler, W.Fiederer, K.Rothermel. *QoS negotiation and resource reservation in distributed Multimedia Systems,*
http://www.informatik.uni-stuttgart.de/ipvr/vs/Publications/Publications.html#1996-rothermel-04, 1996

[83] K.Dünhölter. *Das Web automatisieren mit XML,*
http://members.aol.com/xmldoku, 1998

[84] H. Frystyk, J. Gettys. SMUX Protocol Specification.
http://www.w3.org/TR/1998/WD-mux, 1998

[85] H. Frystyk, J. Gettys. The WebMUX Protocol,
ftp://ftp.ietf.org/internet-drafts/draft-gettys-webmux-00.txt, 1998

[86] H. Frystyk, J. Gettys. HTTP-NG Activity Statement,
http.//www.w3.org/Protocols/HTTP-NG/Activity.html, 1999

[87] H. Frystyk, J. Gettys, B. Janssen, L. Masinter,Larry. Briefing Package for HTTP-NG Project.
http://www.w3.org/Protocols/HTTP-NG/Group/Brief.html, 1997

[88] H. Frystyk, J. Gettys, B. Janssen, M. Spreitzer. HTTP-NG Overview, Problem Statement, Requirements, and Solution Outline
http://www.w3.org/Protocols/HTTP-NG/1998/11/draft-frystyk-httpng-overview-00.txt, 1998

[89] H. Frystyk, B. Janssen, M. Spreitzer. HTTP-ng Architectural Model.
http://www.w3.org/TR/1998/WD-HTTP-NG-architecture, 1998

[90] S. Floyd, J. Mahdavi. TCP-Friendly Unicast Rate-Based Flow Control,
http://www.psc.edu/networking/papers/tcp_friendly.html, 1997

[91] C.Griwodz, M.Liepert, G.On, M.Zink. A Distributed Media Server for the Support of Multimedia teaching. TU Darmstadt, Institut für industrielle Prozeß- und Systemkommunikation.
http://httc.de/medianode/medianode.html, 1999

[92] A.Hagin, G.Dermler, K.Rothermel. Problem Formulations, Models and Algorithms for Mapping Distributed Multimedia Applications to Distributed Computer Systems.
ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-1996-03/TR-1996-03.ps.gz ,1996

[93] P.Halvorsen, T.Plagemann,V.Goebel. *The INSTANCE Project: Operating System Enhancements to Support Multimedia Servers,*
http://confman.unik.no/~paalh/publications/sosp99/halvorsen.html, 1999

[94] J.Heidemann, K. Obraczka, J. Touch. Analysis of HTTP Performance.
http://www.isi.edu/touch/pubs/http-perf96/http-perf96.pdf, 1996

[95] J. Heidemann, K. Obraczka, J. Touch. Modeling the Performance of HTTP Over Several Transport Protocols. http://www.isi.edu/~johnh/PAPERS/Heidemann96a.ps.gz, 1997

[96] J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, L. Stewart.
HTTP Authetication: Basic and Digest Access Authentication,
ftp://ftp.isi.edu/in-notes/rfc2617.txt, RFC 2617, IETF, 1999

[97] Java Media Framework API.
http://www.java.sun.com/marketing/collateral/jmf_ds.html, Sun Microsystems Inc., 1997

[98] B. Janssen. Binary Wire Protocol for HTTP-ng.
http://info.internet.isi.edu/in-drafts/files/draft-janssen-httpng-wire-00.txt, 1998

[99] R. Khare. Building the Perfect Beast - Dreams of a Grand Unified Protocol.
http://www.ics.uci.edu/~rohit/IEEE-L7-applcore.html , 1999

[100] O. Lassila, R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222, W3C, 1999

[101] D. Larner. HTTP-NG Web Interfaces.
http://www.w3.org/TR/1998/WD-HTTP-NG-interfaces-19980710, 1998

[102] [C. Lilley. 10.4 HTTP-NG. http://webbo.enst-bretagne.fr/ActiveWebFr2/eg-uk-tut.book_59.html.
Manchester and North HPC Training & Education Centre, 1995

[103] P. Merle. Integrating Corba Objects within the WWW.
http://www.ansa.co.uk/ANSA/ISF/WWW5/DOWpanel.html, 1996

[104] Microsoft .NET Overview. http://www.microsoft.com/net/default.asp, Microsoft Corporation, 2001

[105] J. C. Mogul, V. N. Padmanabhan. Improving HTTP Latency.
http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/mogul/
HTTPLatency.html, 1994

[106] [D. Veillard. Design of HTTP-NG Testbed.
http://www.w3.org/TR/1998/NOTE-HTTP-NG-testbed-980710, 1998

[107] WAP Forum Ltd.: WAP Architecture.
http://www1.wapforum.org/what/technical/SPEC-WAPArch-19980430.pdf, 1998

[108] Web Developer's: HyperText Transfer Protocol,
http://wdvl.internet.com/Internet/Protocols/HTTP/article.html

[109] J. Whitehead. The Future of Distributed Software Developement on the Internet,
http://www.webtechniques.com/archives/1999/10/whitehead/, 1999

[110] D. Winer. XML-RPC Specification,
http://www.xmlrpc.com/stories/storyReader$7, 1998

[111] D. Winer. XML-RPC for Newbies,
http:/davenet.userland.com/1998/07/14/xmlRpcForNewbies, 1998

[112] D. Veillard. Design of HTTP-NG Testbed,
http://www.w3.org/TR/1998/NOTE-HTTP-NG-testbed-980710, 1998

[113] WAP Forum Ltd.: WAP Architecture,
http://www1.wapforum.org/what/technical/SPEC-WAPArch-19980430.pdf, 1998

[114] Web Developer's: HyperText Transfer Protocol,
http://wdvl.internet.com/Internet/Protocols/HTTP/article.html

[115] J. Walsh. Microsoft spearheads protocol push,
http://archive.infoworld.com/cgi-bin/displayStory.pl?980710.whsoap.htm, 1998

[116] G. Miller, T. Thompson, R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics.
http://www.vbns.net/presentations/papers/MCItraffic.pdf, 1997

[117] M. Nottingham. How (and how not) to Control Caches (1).
http://wdvl.internet.com/Internet/Cache/caches.html, 1999

[118] M. Nottingham. How (and how not) to Control Caches (2),
http://wdvl.internet.com/Internet/Cache/cache2.html, 1999

[119] J. Touch. TCP Control Block Interdependence,
ftp://ftp.isi.edu/in-notes/rfc2140.txt, RFC 2140, IETF, 1997

[120] B. Segal. A Short History of Internet Protocols at CERN, http://wwwinfo.cern.ch/pdp/ns/ben/TCPHIST.html, 1995

[121] A. Silverman. CERN UNIX User Guide, http://consult.cern.ch/writeup/unixguide/main.html, 1996

[122] Spero, Simon E.: Analysis of HTTP Performance Problems, http://www.w3.org/Protocols/HTTP/1.0/HTTPPerformance.html, 1994

[123] Spero, Simon E.: HTTP-NG Architectural Overview, http://www.w3.org/Protocols/HTTP-NG/http-ng-arch.html, 1995

[124] Spero, Simon E.: Next Generation Hypertext Transport Protocol (Draft), http://metalab.unc.edu/ses/ng-notes.txt, 1995

[125] Spero, Simon E.: Progress on HTTP-NG, http://www.w3.org/Protocols/HTTP-NG/http-ng-status.html, 1995

[126] Steven Pemberton et. al. XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0. W3C Recommendation. http://www.w3.org/TR/2000/REC-xhtml1-20000126, W3C, January 2000

[127] QuickTime on the Internet, QuickTime Technology Brief. http://www.apple.com/products, Apple Computers Inc., 1998

[128] IBM Xeena. http://www.alphaworks.ibm.com/aw.nsf/techmain/xeena, IBM Corp. Armonk. NY., 2000

[129] Xerces Java Parser. http://xml.apache.org/xerces-j/index.html, The Apache Software Foundation, August 2001

[130] W. Hodgins et. al. Draft Standard for Learning Object Metadata. New York, NY 10016-5997, USA. http://ltsc.ieee.org/wg12/, IEEE, 2001

[131] ,T. H. Harrison, C. O'Ryan, D. L. Levin, D. C. Schmidt. The Design and Performance of Real-Time CORBA Event Service. http://www.cs.wustl.edu/~schmidt/corba-research.html. Washington University, St. Louis, 1998

[132] J. Postel, J. Reynolds: File Transfer Protocol (FTP). ftp://ftp.isi.edu/in-notes/rfc959.txt, RFC 959, IETF, October 1985

[133] H. Schulzrinne, S. Casner, R. Frederick. RTP: A Transport Protocol for Real-Time Applications. ftp://ftp.isi.edu/in-notes/rfc1889.txt, RFC 1889, IETF, January 1996

[134] S. DeRose, E. Maler, D. Orchard. XML Linking Language (XLink) Version 1.0. W3C Recommendation. http://www.w3.org/TR/2000/REC-xlink-20010627/, W3C, June 2001

[135] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, http://www.w3.org/TR/2000/REC-xml-20001006, W3C, October 2000

[136] Transarc Corporation: AFS Frequently Asked Questions, Transarc Corporation, http://www.angelfire.com/hi/plutonic/afs-faq.html, 1998

[137] Transarc Corporation: AFS User's Guide, Transarc Corporation, http://www.transarc.com/Library/documentation/afs/3.5/unix/user/user.htm, 1999

[138] Leach, P.; Goland, Y.: WebDAV ACL Protocol, http://www.ics.uci.edu/~ejw/authoring/acl/draft-ietf-webdav-acl-00.txt, 1997

[139] Lippert, Lisa: WebDAV Access Control Goals, http://www.ics.uci.edu/~ejw/authoring/acl/draft-ietf-webdav-acl-reqts-00.txt, 1998

[140] Microsoft Standards Activities Team: WebDAV: Evolving the Web into a Read and Write Medium - An Interview with Jim Whitehead, http://msdn.microsoft.com/workshop/standards/webdav.asp, 1999

[141] Faizi, A.; Goland, Y.; Jensen, D.; Whitehead, E.: HTTP Extensions for Distributed Authoring -- WEBDAV, ftp://ftp.isi.edu/in-notes/rfc2518.txt, RFC 2518, IETF, 1999

[142] Spero, Simon E.: SCP - Session Control Protocol V 1.1,
http://metalab.unc.edu/ses/scp.html, 1996

[143] Stein, Greg: DAV Frequently Asked Questions,
http://www.webdav.org/other/faq.html, 1999

[144] Stein, Greg: WebDAV: Distributed Authoring and Versioning,
http://www.webdav.org/papers/dav-adobe.ppt, 1999

[145] University of California Irvine: WebDAV,
http://www.ics.uci.edu/pub/ietf/webdav/intro/webdav_flyer.pdf, 1999

[146] University of Georgia: UNIX System Administration, University Computing and Networking Services,
http://www.uga.edu/~ucns/wsg/unix/sysadmin/

[147] Whitead, Jim: A Brief Introduction to WebDAV,
http://www.ics.uci.edu/~ejw/authoring/intro.html, 1998

[148] J. Wielemaker. SWI-Prolog 4.0 Reference Manual. Department of Social Science Informatics. http://www.swi.psy.uva.nl/projects/SWI-Prolog/Manual/, Amsterdam. 2001.

[149] D. Chamberlin, J. Clark et al.: XQuery 1.0: An XML Query Language. W3C Working Draft, http://www.w3.org/TR/2001/WD-xquery-20010607, W3C, Juni 2001

[150] Eaton, Dave (Ed.): Configuration Management Tools Summary. Freaquently Asked Questions Zusammenfassung aus der Newsgroup im Usenet: comp.software.config-mgmt
URL: http://www.iac.honeywell.com/Pub/Tech/CM/CmTools.html

[151] Meyer, Harald: Versionsverwaltung unter UNIX mit dem Revision Control System (RCS) - eine Einführung. Dokumentation des Rechenzentrums Universität Karlsruhe, 1996
URL: http://www.uni-karlsruhe.de/Uni/RZ/Personen/rz65/RCS-Info/slides.ps.gz

[152] Mosley, Vicky (u.a.): Software Configuration Management Tools: Getting Bigger, Better, and Bolder, 1996
URL: http://stsc.hill.af.mil/crosstalk/1996/jan/cmtools.asp

[153] Zeller, Andreas: Configuration Management with Version Sets - A Unified Software Versioning Model and its Applications. Dissertationsarbeit, Fachbereich für Mathematik und Informatik, Technische Universität Braunschweig, April 1997.
URL: http://www.cs.tu-bs.de/softech/papers/zeller-phd

[154] A.Jonas. Integration von Loss-Collection-RTP(LCRTP) in eine VoD Architektur. Diplomarbeit am Institut für Industrielle Prozeß- und Systemkommunikation an der TU Darmsatdt, 1999

[155] B. Oestereich. Objektorientierte Softwareentwicklung. Oldenbourg Verlag, 1998

[156] T.Plagemann,V.Goebel. *INSTANCE: The Intermediate Storage Node Concept*,
http://www.unik.no/~plageman/ASIAN97-LNCS.ps, 1997

[157] F.Priester. *Entwurf eines Multimedia-Filesystems für Linux*, Diplomarbeit am Institut für Industrielle Prozeß- und Systemkommunikation an der TU Darmsatdt, 1999

[158] R.Ruggaber. Quality of Service (QoS) in verteilten, objektorientierten Systemen.
http://www.telematik.informatik.uni-karlsruhe.de/forschung/klung98/node18.html,2000

[159] D.Rusling. The Linux Kernel. URL:http://compy.ww.tu-berlin.de/KERNEL-HOWTO/ ,2000

[160] L.Schreier, M.Brown, K.Finn, B.Sabata. MuX: Distributed Multimedia I/O Server.
www.erg.sri.com/people/sabata/projects/distmm.html ,1994

[161] J.Schneider. Implementierung eines RTSP_Servers. Diplomarbeit am Institut für Industrielle Prozeß- und Systemkommunikation an der TU Darmsatdt, 1999

[162] P.J.Shenoy, P.Goyal, H.M.Vin. *Issues in Multimedia Server Design*, in: ACM Computing Surveys, Vol. 27, No. 4, Seiten 636-639, Dezember 1995,
http://www.cs.utexas.edu/users/dmcl/papers/ps/ComputingSurvey.ps

[163] Tewari, Renu. Architectures and Algorithms for Scalable Wide-area Information Systems. Dissertation am Distributed Multimedia Computing Lab der Universtät Austin/Texas, http://www.cs.utexas.edu/users/dmcl/papers/dis/renu.ps,1998

[164] F.A.Tobagi, S.-H. Gary Chan. *1999 Status Report on "Scalable On-Demand Video Services" Project,* http://pocari.stanford.edu/telecom/resproj/distributedserverdescpage.html, 1999

[165] F.A.Tobagi, D.Harris. *Distributed Server Networks and the Delivery of Scalable Multimedia Services - Project description,* http://pocari.stanford.edu/telecom/resproj/distributedserverdescpage.html, 1998

[166] R.Tolksdorf. *XML und darauf basierende Standards: Die neuen Auszeichnungssprachen des Web,* in: Informatik Spektrum, Ausgabe 12/1999, S.407 - 421

[167] H.M.Vin, A.Goyal, P.Goyal. IAlgorithms for Designing Multimedia Servers. 1995, http://www.cs.utexas.edu/users/dmcl/papers/ps/CompComm95.ps

[168] Zimmer, Mirko. XML-Eine kurze Einführung. http://userpage.fu-berlin.de/~mizi/html/xml1.html,2000

Anhang B - Abkürzungen

ACE	Access Control Entry
ACL	Access Control List
AFS	Andrew File System
AVO	Audio Visual Object
AWT	Abstract Windowing Toolkit
CAD	Computer Aided Design
CINEMA	Configurable Integrated Multimedia Architecture
CM	Configuration Management
CORBA	Common Object Request Broker Architecture
CSCW	Computer Supported Cooperative Work
CSS	Cascading Style Sheet
CVS	Concurrent Versions System
DASL	DAV Searching and Locating
DAV	Distributed Authoring and Versioning
DB	Datenbank
DBMS	Data Base Management System
DMIF	Delivery Multimedia Integration Framework
DMS	Document Management System
DOM	Document Object Model
DTD	Document Type Definition
FTP	File Transfer Protocol
GUI	Graphical User Interface
HTML	Hypertext Markup Language
httc	Hessisches Telemedia Technologie Kompetenz-Center
HTTP	HyperText Transfer Protocol
HTTP-NG	HyperText Transfer Protocol, Next Generation
HyNoDe	Hypertext News on Demand

IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IIOP	nternet Inter-ORB Protocol
IMAP	Interactive Mail Access Protocol
IP	Internet Protocol
ISDN	Integrated Servcies Digital Network
ISO	International Organization for Standardization
JMF	Java Media Framework
MIME	Multi-purpose Internet Mail Extension
MPEG	Moving Pictures Expert Group
NNTP	Network News Tranfer Protocol
NoD	News on Demand
ORB	Object Request Broker
POI	Point of Information
POP	Post Office Protocol
POS	Point of Sale
QoS	Quality of Service
RCS	Revision Control System
RED	Random Early Detection
RPC	Remote Procedure Call
RTT	Round Trip Time
SCCS	Source Code Control System
SCM	Software Configuration Management
SGML	Standard Generalized Markup Language
SMTP	Simple Mail Transfer Protocol
SRM	Session and Resource Management
SSL	Secure Socket Layer
TCP	Transfer Control Program
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium

WebDAV	DAV im WWW
WWW	World Wide Web
XHTML	Extensible HTML
XLink	XML Linking Language
XML	Extensible Markup Language
XPointer	XML Pointer Language
XSL	XML Stylesheet Language

Anhang C - Zur Datenstruktur

Im folgenden wird die Datentyp-Definition des Inhalts einer verteilten Infrastruktur zum Austausch multimedialen Vorlesungsmaterials (vgl. Abschnitt 2.1.3) präsentiert. Diese in Kapitel 4 entwickelte Struktur liegt den Erweiterungen der Kapitel 5, 6 und 7 zugrunde.

```
<!--

  PUBLIC "-// KOM TU Darmstadt//DTD medianode 1.0//EN"
  SYSTEM "http://www.kom.e-technik.tu-darmstadt.de/~lipi/dtd/medianode-1.0.dtd"

-->

<!ENTITY % datetype "(#PCDATA)">

<!ENTITY % stringtype "(#PCDATA)">

<!ENTITY % encoding "(None|Base64|QuotedPrintable|X-uuencode)">

<!ENTITY % resource.structure "(sequence|alternative)">

<!ENTITY % persistence "(explicit|temporary)">
```

Kürzung:

An dieser Stelle sind aus Gründen der Übersichtlichkeit aus XML- und LOM-Spezifikation übernommene Entity-Definitionen (*language*, *location* und *mime-type*) nicht abgedruckt.

```
<!ELEMENT mediarug (region,devices.list)>

<!ELEMENT region (metadata.region,(region*|department))>
<!ELEMENT metadata.region (principals.list)>
<!ATTLIST metadata.region
      name CDATA #REQUIRED
      owner.ref IDREF #REQUIRED
      id ID #IMPLIED
      admin.refs IDREFS #IMPLIED
>

<!ELEMENT principals.list (principal)*>
<!ELEMENT principal (user|group)>
<!ATTLIST principal
      id ID #IMPLIED
      name CDATA #REQUIRED
      comment CDATA #IMPLIED
>
<!ELEMENT user (PersonScheme)>
<!ELEMENT group EMPTY>
<!ATTLIST group
      principal.refs IDREFS #IMPLIED
```

```
>

<!ELEMENT devices.list (device)*>
<!ELEMENT device (dimension)+>
<!ATTLIST device
      name CDATA #REQUIRED
      id ID #IMPLIED
      mimetype %mimetype; #IMPLIED
>

<!ELEMENT dimension EMPTY>
<!ATTLIST dimension
      id ID #IMPLIED
      name CDATA #REQUIRED
>

<!ELEMENT department (resources.tree,styles.list,presentations.list,machines.list)>

<!ELEMENT resources.tree (resource)*>
<!ELEMENT resource (metadata.resource?,resource.reference*,(resource*|blob))>
<!ATTLIST resource
      id ID #IMPLIED
      resource.structure %resource.structure; "sequence"
      bandwidth CDATA "0"
>
<!ELEMENT blob (#PCDATA)>
<!ATTLIST blob
      id ID #IMPLIED
      source.url CDATA #IMPLIED
      mimetype %mimetype; #REQUIRED
      encoding %encoding; #REQUIRED
>
<!ELEMENT resource.reference EMPTY>
<!ATTLIST resource.reference
      substitution.string CDATA #IMPLIED
      resource.ref IDREF #REQUIRED
>

<!ELEMENT styles.list (style)*>
<!ELEMENT style (metadata.resource)>
<!ATTLIST style
      resource.ref IDREF #IMPLIED
      id ID #IMPLIED
      name CDATA #REQUIRED
>




<!ELEMENT presentations.list (presentation)*>
<!ELEMENT presentation (metadata.resource?,presentation*)>
<!ATTLIST presentation
      id ID #IMPLIED
      resource.ref IDREF #IMPLIED
      style.ref IDREF #IMPLIED
      persistence %persistence; #IMPLIED
      step CDATA #IMPLIED
      title CDATA #IMPLIED
>

<!ELEMENT metadata.resource (device.amount*,description?,Date?,Title,author?,owner)>
<!ELEMENT device.amount EMPTY>
```

```
<!ATTLIST device.amount
      dimension.ref IDREF #REQUIRED
      amount CDATA #IMPLIED
>
<!ELEMENT description (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Title (LangStringScheme)>
<!ELEMENT LangStringScheme (Locale,String)>
<!ELEMENT Locale (LocaleScheme)>
<!ELEMENT String (#PCDATA)>
<!ELEMENT LocaleScheme (Location,HumanLanguage)>
<!ELEMENT Location EMPTY>
<!ATTLIST Location
      location %location; "ge"
>
<!ELEMENT HumanLanguage EMPTY>
<!ATTLIST HumanLanguage
      language %language; "en"
>
<!ELEMENT author (Person)>
<!ELEMENT owner (Person)>
<!ELEMENT machines.list (machine)*>
<!ELEMENT machine (metadata.resource,medianode*)>
<!ELEMENT medianode (access.session*,verifier.session*,storage.session*)>
<!ELEMENT access.session EMPTY>
<!ATTLIST access.session
      implementation.ref IDREF #IMPLIED
>
<!ELEMENT verifier.session EMPTY>
<!ATTLIST verifier.session
      implementation.ref IDREF #IMPLIED
>
<!ELEMENT storage.session EMPTY>
<!ATTLIST storage.session
      implementation.ref IDREF #IMPLIED
>
<!ELEMENT Person (PersonScheme|user.reference)>
<!ELEMENT user.reference EMPTY>
<!ATTLIST user.reference
      user.ref IDREF #REQUIRED
>
<!ELEMENT PersonScheme
(FirstName?,MiddleName?,PrefixToLastName?,LastName,Affiliation?,Identifier?)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT MiddleName (#PCDATA)>
<!ELEMENT PrefixToLastName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT Affiliation (OrganizationScheme)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT OrganizationScheme (Name,Address?)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
```

Anhang D - Zum Rollenkonzept

In diesem Anhang wird eine einfache Prototypimplementierung für das Rollenkonzept aus Kapitel 6 vorgestellt.

Die folgenden Dateien beschreiben die allgemeine Semantik und einen Beispiel-Inhalt in der Programmiersprache Prolog.

D.1 Ableitung der Rollen-Zugriffsrechte

Der folgende Code implementiert innerhalb eines Moduls für SWI-Prolog [148] die Ableitung der Rollenzugriffsrechte für das vorgeschlagene Rollenkonzept. Durch Prolog-Anfragen können einzelne Zugriffsmöglichkeiten getestet werden, aber auch Listen von möglichen Zugriffen generiert werden.

Konkreter Inhalt muss durch Prolog-Prädikate abgefragt werden können. Dies kann durch explizite Formulierung als Prolog-Fakten geschehen oder durch dynamische Abfragen auf externen Daten. Im folgenden Abschnitt ist ein Beispieldatensatz gegeben.

```
% generated: 20 February 2001
%
%
%   rights: for an XML based Distributed Multimedia Archive
%
%   (c) 2001, KOM TU Darmstadt
%

:- module(rights
   , [access/3,principal/2,action/1,grant/1]
     ).



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% now, eval:

% facts/decisions:
action(write).
action(read).
action(administer).
action(delete).
action(createChild).
action(deleteChild).

grant(grant).
grant(deny).

% Principal definition: local to node
principal(NODE, user(ANCESTOR, USER)) :-
   content(CONTENT),
   principal(CONTENT, NODE, user(ANCESTOR, USER)).
principal(CONTENT, NODE, user(ANCESTOR, USER)) :-
```

```
   ancestorOrSelf(CONTENT, ANCESTOR, NODE),
   CONTENT:user(ANCESTOR, USER).
principal(CONTENT, NODE, group(ANCESTOR, GROUP)) :-
   ancestorOrSelf(CONTENT:ANCESTOR, NODE),
   CONTENT:group((ANCESTOR, GROUP), _).


% access without roles:
access(NODE, (PRINCIPAL, ROLE), ACTION) :-
   content(CONTENT),
   access(CONTENT, NODE, (PRINCIPAL, ROLE), ACTION).
access(CONTENT, NODE, (PRINCIPAL, nil), ACTION) :-
   principal(CONTENT, NODE, PRINCIPAL),
   action(ACTION),
   treeacl(CONTENT, NODE,PRINCIPAL,grant,ACTION),
   not(treeacl(CONTENT, NODE,PRINCIPAL,deny,ACTION)).

% acces with roles:
access(CONTENT, NODE, (PRINCIPAL, ROLE), ACTION) :-
   treeRole(CONTENT, NODE, ROLE),
   principal(CONTENT, NODE, PRINCIPAL),
   access(CONTENT, NODE, (PRINCIPAL, nil), ACTION),
   treeRole(CONTENT, NODE, ROLE),
   roleAcl(CONTENT, NODE, (PRINCIPAL, ROLE), ACTION).


% search for active ACLs
treeacl(NODE,PRINCIPAL,GRANT,ACTION) :-
   content(CONTENT),
   treeacl(CONTENT, NODE,PRINCIPAL,GRANT,ACTION).
treeacl(CONTENT, NODE,PRINCIPAL,GRANT,ACTION) :-
   principal(CONTENT, NODE, PRINCIPAL),
   grant(GRANT),
   action(ACTION),
   (
       localAcl(CONTENT, NODE,PRINCIPAL,GRANT,ACTION) ;
       (
      ancestor(CONTENT, NODE2, NODE),
      treeacl(CONTENT, NODE2,PRINCIPAL,GRANT,ACTION)
       )
   ),
   !.


racl(CONTENT, NODE, WHO, GRANT, HOW) :-
   action(HOW),
   CONTENT:racl(NODE, WHO, GRANT, all).
racl(CONTENT, NODE, WHO, GRANT, HOW) :-
   CONTENT:racl(NODE, WHO, GRANT, HOW).


% principal access to nodes
% special rules:
localAcl(NODE, PRINCIPAL, GRANT, ACTION) :-
   content(CONTENT),
   localAcl(CONTENT, NODE, PRINCIPAL, GRANT, ACTION).
localAcl(CONTENT, NODE, PRINCIPAL, GRANT, ACTION) :-
   action(ACTION),
   (
       CONTENT:acl(NODE,PRINCIPAL,GRANT,ACTION) ;
       CONTENT:acl(NODE,PRINCIPAL,GRANT,all)
   ).
```

```
localAcl(CONTENT, NODE, user(DOMAIN, USER), grant, administer) :-
   CONTENT:node(NODE),
   CONTENT:owner(NODE, user(DOMAIN, USER)),
   ancestorOrSelf(CONTENT, NODE, DOMAIN).
% global rules
localAcl(CONTENT, NODE, PRINCIPAL, GRANT, ACTION) :-
   CONTENT:group(GROUP, PRINCIPAL),
   localAcl(CONTENT, NODE,GROUP,GRANT,ACTION).

roleAcl(NODE, (PRINCIPAL, ROLE), ACTION) :-
   content(CONTENT),
   roleAcl(CONTENT, NODE, (PRINCIPAL, ROLE), ACTION).
% role access to node types:
roleAcl(CONTENT, NODE, (PRINCIPAL, ROLE), ACTION) :-
   CONTENT:node(NODE),
   CONTENT:type(NODE, NODETYPE),
   action(ACTION),
   not(racl(CONTENT, NODETYPE, ROLE, deny, ACTION)),
   (
       localAcl(CONTENT, NODE, PRINCIPAL, grant, ACTION) ;
       (
      CONTENT:parent(PARENT, NODE),
      roleAcl(CONTENT, PARENT, (PRINCIPAL, ROLE), ACTION),
      roleAcl(CONTENT, relation(PARENT, NODE), (PRINCIPAL, ROLE), ACTION)
       )
   ),
   !.
% access to relation types:
roleAcl(CONTENT, relation(NODE1, NODE2), (_, ROLE), ACTION) :-
   CONTENT:parent(NODE1, NODE2),
   CONTENT:type((NODE1, NODE2), RELATIONTYPE),
   not(racl(CONTENT, RELATIONTYPE, ROLE, deny, ACTION)).




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Semantic Checks:
correct :-
   content(CONTENT),
   correct(CONTENT).
correct(CONTENT) :-
   incorrect(CONTENT),
   !,
   msgError(['List of all errors is:\n']),
   incorrect(CONTENT),
   fail.
correct(_).


% every node has at MOST 1 owner
incorrect(CONTENT) :-
   CONTENT:node(NODE),
   CONTENT:owner(NODE, OWNER1),
   CONTENT:owner(NODE, OWNER2),
   not(OWNER1 = OWNER2),
   msgError(['node %t has two owners: %t != %t', NODE, OWNER1, OWNER2]).

% every node has at 1 owner on its way to root:
incorrect(CONTENT) :-
   CONTENT:node(ROOT),
   not(CONTENT:parent(_, ROOT)),
```

```
   not(CONTENT:owner(ROOT, _)),
   msgError([’root %t node has no owner’, ROOT]).

% there’s exactly one root
incorrect(CONTENT) :-
   CONTENT:node(ROOT1),
   not(CONTENT:parent(_, ROOT1)),
   !,
   CONTENT:node(ROOT2),
   not(CONTENT:parent(_, ROOT2)),
   not(ROOT1 = ROOT2),
   msgError([’there are two roots %t and %t’, ROOT1, ROOT2]).

% every user/role in an (r)ACL must be defined at the point of an (r)ACL...
incorrect(CONTENT) :-
   CONTENT:acl(NODE, WHO, _, _),
   not(treePrincipal(CONTENT, NODE, WHO)),
   msgError([’user %t in ACL of node %t not defined’, WHO, NODE]).
incorrect(CONTENT) :-
   CONTENT:racl(NODE, ROLE, _, _),
   not(treeRole(CONTENT, NODE, ROLE)),
   msgError([’role %t in rACL of node %t not defined’, ROLE, NODE]).


% groups can only contain users:
% groups can only contain users known at the place of the group:
incorrect(CONTENT) :-
   group(group(DOMAIN, GROUP), user(USERNODE, USER)),
   not(call(CONTENT:user(USERNODE, USER))),
   msgError([’user %t in group %t at node %t not existent’, user(USERNODE, USER),
GROUP, DOMAIN]).
incorrect(CONTENT) :-
   group(group(DOMAIN, GROUP), user(USERNODE, USER)),
   not(ancestorOrSelf(CONTENT, USERNODE, DOMAIN)),
   msgError([’user %t in group %t at node %t not visible here’, user(USERNODE, USER),
GROUP, DOMAIN]).







%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% helper predicates

ancestor(NODE2, NODE) :-
   content(CONTENT),
   ancestor(CONTENT, NODE2, NODE).

ancestor(CONTENT, NODE2, NODE) :-
   CONTENT:parent(NODE2, NODE).
ancestor(CONTENT, NODE2, NODE) :-
   CONTENT:parent(NODE3, NODE),
   ancestor(CONTENT, NODE2, NODE3).

ancestorOrSelf(NODE, NODE) :-
   content(CONTENT),
   ancestorOrSelf(CONTENT, NODE, NODE).
ancestorOrSelf(CONTENT, NODE, NODE) :-
```

```
   CONTENT:node(NODE).
ancestorOrSelf(CONTENT, NODE2, NODE) :-
   ancestor(CONTENT, NODE2, NODE).


treeRole(NODE, role(ANCESTORORSELF, ROLE)) :-
   content(CONTENT),
   treeRole(CONTENT, NODE, role(ANCESTORORSELF, ROLE)).
treeRole(CONTENT, NODE, role(ANCESTORORSELF, ROLE)) :-
   ancestorOrSelf(CONTENT, ANCESTORORSELF, NODE),
   CONTENT:role(ANCESTORORSELF, ROLE).

treePrincipal(NODE, PRINCIPAL) :-
   content(CONTENT),
   treeRole(CONTENT, NODE, PRINCIPAL).
treePrincipal(CONTENT, NODE, user(ANCESTORORSELF, USER)) :-
   ancestorOrSelf(CONTENT, ANCESTORORSELF, NODE),
   CONTENT:user(ANCESTORORSELF, USER).
treePrincipal(CONTENT, NODE, group(ANCESTORORSELF, USER)) :-
   ancestorOrSelf(CONTENT, ANCESTORORSELF, NODE),
   CONTENT:group(ANCESTORORSELF, USER).

msgError([FORMAT | ARGS ]) :-
   tell(user_error),
   WRITEF =.. [ writef, FORMAT, ARGS ],
   call(WRITEF),
   nl,
   told.




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% these are for testing:

delAcl(ME, acl(NODE, PRINCIPAL, GRANT, ACTION)) :-
   content(CONTENT),
   addAcl(CONTENT, ME, acl(NODE, PRINCIPAL, GRANT, ACTION)).
addAcl(CONTENT, ME, acl(NODE, PRINCIPAL, GRANT, ACTION)) :-
   grant(GRANT),
   access(CONTENT, NODE, ME, administer),
   not(localAcl(CONTENT, NODE, PRINCIPAL, GRANT, ACTION)),
   assert(CONTENT:acl(NODE, PRINCIPAL, GRANT, ACTION)).
addAcl(CONTENT, ME, acl(NODE, PRINCIPAL, GRANT, ACTION)) :-
   grant(GRANT),
   action(ACTION),
   localAcl(CONTENT, NODE, PRINCIPAL, GRANT, ACTION),
   access(CONTENT, NODE, ME, administer),
   retract(CONTENT:acl(NODE, PRINCIPAL, GRANT, ACTION)).

delNode(ME, NODE) :-
   content(CONTENT),
   delNode(CONTENT, ME, NODE).
delNode(CONTENT, ME, NODE) :-
   access(CONTENT, NODE, ME, delete),
   deleteParentRelation(CONTENT, ME, NODE, [CONTENT:node(NODE)], FACTS1),
   deleteChildren(CONTENT , ME, NODE, FACTS1, FACTS2),
   retractList(FACTS2).

deleteParentRelation(ME, CHILD, FACTSIN, [ CONTENT:parent(PARENT, CHILD) | FACTSIN ])
:-
   content(CONTENT),
```

```
   deleteParentRelation(CONTENT, ME, CHILD, FACTSIN, [ CONTENT:parent(PARENT, CHILD)
| FACTSIN ]).
deleteParentRelation(CONTENT, ME, CHILD, FACTSIN, [ CONTENT:parent(PARENT, CHILD) |
FACTSIN ]) :-
   CONTENT:parent(PARENT, CHILD),
   access(CONTENT, PARENT, ME, deleteChild).
deleteParentRelation(CONTENT, _, ROOT, FACTSIN, FACTSIN) :-
   not(CONTENT:parent(_, ROOT)).




addParentRelation(ME, PARENT, CHILD, FACTSIN, [ parent(PARENT, CHILD) | FACTSIN]) :-
   content(CONTENT),
   addParentRelation(CONTENT, ME, PARENT, CHILD, FACTSIN, [ parent(PARENT, CHILD) |
FACTSIN]).
addParentRelation(CONTENT, ME, PARENT, CHILD, FACTSIN, [ parent(PARENT, CHILD) |
FACTSIN]) :-
   CONTENT:node(PARENT),
   CONTENT:node(CHILD),
   not(PARENT = CHILD),
   not(CONTENT:parent(_, CHILD)),
   not(CONTENT:ancestor(CHILD, PARENT)),
   access(CONTENT, PARENT, ME, createChild).



lookup(ME, NODE) :-
   CONTENT:node(NODE),
   lookup(CONTENT, ME, NODE).
lookup(CONTENT, ME, NODE) :-
   access(CONTENT, NODE, ME, read),
   write(CONTENT:lookup(ME, NODE)), nl,
   lookupAllChildren(CONTENT, ME, NODE).

lookupAllChildren(ME, NODE) :-
   content(CONTENT),
   lookupChildren(CONTENT, ME, NODE).
lookupAllChildren(CONTENT, ME, NODE) :-
   CONTENT:parent(NODE, CHILD),
   lookupRelation(CONTENT, ME, (NODE, CHILD)),
   lookup(CONTENT, ME, CHILD),
   fail.
lookupAllChildren(_, _, _).

lookupRelation(PRINCIPAL, (NODE, CHILD)) :-
   content(CONTENT),
   lookupRelation(CONTENT, PRINCIPAL, (NODE, CHILD)).
lookupRelation(CONTENT, PRINCIPAL, (NODE, CHILD)) :-
   principal(CONTENT, NODE, PRINCIPAL),
   CONTENT:parent(NODE, CHILD),
   CONTENT:type((NODE, CHILD), RELATIONTYPE),
   write(CONTENT:RELATIONTYPE : NODE -> CHILD).
% role-restricted lookup
lookupRelation(CONTENT, (PRINCIPAL, ROLE), (NODE, CHILD)) :-
   principal(CONTENT, NODE, PRINCIPAL),
   treeRole(CONTENT, NODE, ROLE),
   CONTENT:parent(NODE, CHILD),
   CONTENT:type((NODE, CHILD), RELATIONTYPE),
   not(racl(CONTENT, RELATIONTYPE, ROLE, deny, read)),
   write(CONTENT:RELATIONTYPE : NODE -> CHILD).
```

D.2 Darstellung der Metadaten

Die folgende Datei formuliert in Form von Prologfakten einen Datenbestand bestehend aus Inhalts-, Benutzer- und Systemmetadaten für die Ableitung der Rollenzugriffsrechte im vorigen Abschnitt.

Die Datei enthält auch eine explizite Abbildung auf Objekt- und Relations-Typen. Diese Information könnte beim Parsen einer entsprechenden Repräsentation aus einem formal definierten, hierarchischen Datentyp (etwa einem XML-Dokument) dynamisch gewonnen werden.

```
% generated: 20 February 2001
%
%
%   medianode definitions
%
%   Author: lipi
%   (c) 2001, KOM TU Darmstadt
%
%   updated: 20 February 2001
%


:- module(medianode,
   [xml_content/1,
   xml_public_dtd/1,
   node/1,
   parent/2,
   type/2,
   user/2,
   group/2,
   role/2,
   acl/4,
   racl/4]
    ).


% declare file
% xml_content('/afs/kom/home/lipi/share/yunus/xml-files/medianode-init.med').
xml_content('medianode-small.med').
xml_public_dtd('-// KOM TU Darmstadt//DTD medianode 2.0 pre 5//EN').

% load rest of content via DOM and XML-file!!!
:-
   [dom],
   dom:load_content_module(medianode).


% interprete DOM information:

node(ID) :-
   dom:dom_node(ID).
parent(FATHER, CHILD) :-
   dom:dom_parent(FATHER, CHILD).
type(ID, TYPE) :-
   dom:dom_type(ID, TYPE).
type((FATHER, CHILD), RELATIONTYPE) :-
   dom:dom_type((FATHER, CHILD), RELATIONTYPE).
```

```
% additional definitions:
% THESE ARE NOT YET IN THE DB, therefore: fake user definitions...


user(mediarug, system).
user(mediarug, griff).
user(kom,lipi).
user(kom,rst).
user(fhg,enc).

group(group(mediarug, admin), user(mediarug, griff)).
group(group(kom,staff),user(kom,lipi)).
group(group(kom,staff),user(kom,rst)).


acl(mediarug, user(mediarug, system), grant, all).
acl(mediarug, user(mediarug, griff), grant, administer).
acl(mediarug, user(mediarug, griff), grant, createChild).
acl(kom, user(mediarug, griff), deny, all).
acl(fhg, user(mediarug, griff), deny, all).
acl(kom, group(kom, staff), grant, all).
acl(kom, user(kom, rst), deny, administer).
acl(fhg, user(fhg, enc), grant, all).


% role definition
role(mediarug, system).
role(kom, admin).
role(kom, author).
role(fhg, author).

racl(resource, role(mediarug, system), deny, write).
racl(mediarug_location, role(mediarug, admin), deny, write).
racl(mediarug_location, role(mediarug, admin), deny, admin).
racl(department, role(kom, author), deny, write).
racl(resource, role(fhg, author), deny, admin).
```

Anhang E - Zur Versionierung

Beispielhaft wurde das versionierte Auslesen, Erstellen und Aktualisieren einer Ressource mittels Datenstrukturtransformationen umgesetzt. Die Transformationen der privaten und öffentlichen Datenbanken wurden mit der Extensible Style Sheet Language (XSL) auf dem in XML formulierten Demonstrations-Datenbestand des Medianode-Projektes auf der CeBIT 2000 in Hannover durchgeführt.

E.1 Prototypische Anfragesprache

Um die entsprechenden Befehle zu spezifizieren, wurde hilfsweise eine Kommandosprache in XML modelliert, welche im folgenden kurz umrissen werden soll. Die Document Type Definition (DTD) dieser Sprache fängt mit folgenden Elementedeklarationen an:

```
<!ELEMENT Request (user,command)>
<!ELEMENT user EMPTY>
<!ATTLIST user
      user.id CDATA #REQUIRED
>
<!ELEMENT command (insertnew|delete|change|retrieve)>
```

Die einzelnen Kommandos enthalten Angaben über die zu aktualisierende Ressource, den neuen Inhalt der Ressource, die neuen Versionsnummern etc.. Die Kommandosprache enthält alle Angaben zur Aktualisierung einer Ressource. Diese als XML-Text spezifizierte Sprache enthält als Wurzelelement *Request.* Dieses Element enthält ein Element *user* und *command*. Das Element *user* hat als Attribut die *user.id*, welches den Nutzer identifiziert, der die Aktualisierung durchführen will. Dem folgt das Element *command*, welches entweder die Kommandoelemente *insertnew, delete*, *change* oder *retrieve* enthält.

Zur Durchführung etwa einer Update-Operation muss ein entsprechendes Kommando zunächst in der Kommandosprache formuliert werden. Dieses Kommando wird in mehreren Schritten ausgeführt, die jeweils als XSL-Transformationen ausgeführt sind. Es resultiert ein neuer Datenbestand, der sich nur durch das Ergebnis der gewünschten Operation unterscheidet. Erzeugt wird auch sowohl die zusätzliche Versionierungsinformation innerhalb des neuen Bestandes wie auch die entsprechend notwendigen Notifikationen an die Nutzer, es werden aber keine der spezifierten Interaktionen mit dem Nutzer (etwa zur Auswahl einer Variante als aktuell) vorgenommen.

Eine Kommandofdatei sieht dabei beispielhaft folgendermassen aus und enthält alle Angaben, die in die Versionsknoten eingefügt werden müssen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Request>
  <user user.id="maier"/>
  <command>
    <insertnew newresource.id=""
```

```
      newversion.id="" oldresource.id="resource.kom.0"
      oldversion.id="1.0" version.date="2001-01-10"
      version.lockeddate="2001-01-10" version.lockstate="unlocked">
      <resource id="" physical.location="" resource.structure="sequence">
        <blob encoding="plain-text" id="" mime.type="text/html"
source.url="http://www.beispiel.de"/>
      </resource>
    </insertnew>
  </command>
</Request>
```

E.2 Typ-Definition der Versionierungsdaten

DTD der im Prototyp verwendeten Anfragesprache:

Dies ist die DTD für die Kommandosprache zum Update und Retrieval der Ressourcen aus der öffentlichen in die private Datenbank. Dabei entspricht die Ressourcenstruktur in dieser DTD genau der Struktur der Ressourcenknoten in ohne Versionsinformationen. In der Demonstration wurde allerdings nur ein Ressourcenknoten eingelesen und aktualisiert.

```
<!ENTITY % version.lockstate "(unlocked | nowrite | noread)">
<!ELEMENT Request (user,command)>
<!ELEMENT user EMPTY>
<!ATTLIST user
      user.id CDATA #REQUIRED
>
<!ELEMENT command (insertnew|delete|change|retrieve)>
<!ELEMENT insertnew (resource)?>
<!ATTLIST insertnew
      oldversion.id CDATA #IMPLIED
      newversion.id CDATA #IMPLIED
      oldresource.id CDATA #IMPLIED
      version.lockstate (unlocked|nowrite|noread) "unlocked"
      version.date CDATA #IMPLIED
      version.lockeddate CDATA #IMPLIED
>
<!ELEMENT delete EMPTY>
<!ATTLIST delete
      oldversion.id CDATA #IMPLIED
      newversion.id CDATA #IMPLIED
      oldresource.id CDATA #IMPLIED
      deleted.resource.id CDATA #IMPLIED
      version.lockstate CDATA #IMPLIED
      version.date CDATA #IMPLIED
      version.lockeddate CDATA #IMPLIED
>
<!ELEMENT change (resource*)?>
<!ATTLIST change
      oldversion.id CDATA #IMPLIED
      newversion.id CDATA #IMPLIED
      oldresource.id CDATA #IMPLIED
      version.lockstate (unlocked|nowrite|noread) "unlocked"
      version.date CDATA #IMPLIED
      version.lockeddate CDATA #IMPLIED
>
<!ELEMENT retrieve EMPTY>
<!ATTLIST retrieve
```

```
      resource.id CDATA #IMPLIED
     version.id CDATA #IMPLIED
>

<!ELEMENT resource (metadata.resource?,multimedia.reference*,(resource*,blob))>
<!ATTLIST resource
      id CDATA #IMPLIED
      resource.structure (sequence|alternative) "sequence"
      physical.location CDATA #IMPLIED
>
<!ELEMENT multimedia.reference (resource.ref)*>
<!ATTLIST multimedia.reference
      substitution.string CDATA #IMPLIED
>
<!ELEMENT resource.ref EMPTY>
<!ATTLIST resource.ref
      referenced.id CDATA #IMPLIED
      referenced.version CDATA #IMPLIED
      reference.type CDATA #IMPLIED
      branch.no CDATA #IMPLIED
>
<!ELEMENT blob (#PCDATA)>
<!ATTLIST blob
      id CDATA #IMPLIED
      source.url CDATA #IMPLIED
      mime.type CDATA #IMPLIED
      encoding CDATA #IMPLIED
>
<!ELEMENT metadata.resource (device.amount*,description,Date,Title,author,owner)>
<!ELEMENT device.amount EMPTY>
<!ATTLIST device.amount
      dimension.ref CDATA #IMPLIED
      amount CDATA #IMPLIED
>
<!ELEMENT description (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Title (LangStringScheme)>
<!ELEMENT author (Person)>
<!ELEMENT owner (Person)>
<!ELEMENT LangStringScheme (Locale,String)>
<!ELEMENT Locale (Localscheme)>
<!ELEMENT String (#PCDATA)>
<!ELEMENT Localscheme (Location,Humanlanguage)>
<!ELEMENT Location EMPTY>
<!ATTLIST Location
      location CDATA #IMPLIED
>
<!ELEMENT Humanlanguage EMPTY>
<!ATTLIST Humanlanguage
      language CDATA #IMPLIED
>
<!ELEMENT Person (Personscheme,user.reference)>
<!ELEMENT Personscheme
(Firstname?,Middlename?,PrefixToLastName?,Lastname,Affiliation?,Identifier?)>
<!ELEMENT user.reference EMPTY>
<!ATTLIST user.reference
      user.ref CDATA #IMPLIED
>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Middlename (#PCDATA)>
<!ELEMENT PrefixToLastName (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
```

```
<!ELEMENT Affiliation (Organizationscheme)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT Organizationscheme (Name,Adress)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Adress (#PCDATA)>
```

DTD der versionierten Daten:

Hier werden die relevanten, versionsbezogenen Ausschnitte des Datentyps der Datenstruktur für die öffentliche mit Versionsinformationen behaftete Datenbank vorgestellt. Sie bildet die Erweiterung der Datenstruktur, die in Kapitel 4 festgelegt wurde.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- changes 2001-03-31 YK:
  PUBLIC "-// KOM TU Darmstadt//DTD medianode 2.0 pre 5//EN"
  SYSTEM "http://www.kom.e-technik.tu-darmstadt.de/~lipi/dtd/medianode-2.0pre5.dtd"
-->

…

** TYPE ENTITIES:
-->

<!ENTITY % notification.type "(push|pull)">

<!ENTITY % notification.form "(email|flag)">

<!ENTITY % recent.updates "(Yes|NO)">

…
<!--
================================================================================
==========
*** ATTRIBUTE DEFINITION ENTITIES:
-->

<!ENTITY % updatenotification.attr "
     resource.id ID #REQUIRED
     version.id ID #REQUIRED
     newversion.id ID #REQUIRED
     update.by ID #REQUIRED
     update.date ID #REQUIRED
">

…
<!ENTITY % versionneighbour.attrs "
      referenced.id IDREF #REQUIRED
      referenced.version CDATA #REQUIRED
">

<!ENTITY % reference.attrs "
      %versionneighbour.attrs;
      reference.type %reference.type;
">



…
<!-- user context description: when, where, which content -->
```

```
<!ELEMENT user.context EMPTY>
<!ATTLIST user.context
      machine %URI; #IMPLIED
      timestamp CDATA #IMPLIED
      %reference.attrs;
      number CDATA #REQUIRED
>
<!-- history is a list of contexts -->
<!ELEMENT user.history (user.context*)>

<!---Here the notifications for the users are stored----->
<!ELEMENT notification (updates.list)>
<!ATTLIST notification
    %notification.type; #REQUIRED "push"
    %notification.type; #REQUIRED "email"
    %recent.updates; #REQUIRED "No"
>

<!ELEMENT  updates.list (update*)>
>
<!ELEMENT update (#PCDATA)
>
<!ATTLIST update
   %updatenotification.attr; #REQUIRED
>

<!--
*** the resources tree: resource elements are nodes and leaves, while
          leaves contain exactly one blob
-->

<!ELEMENT resources.list (resource)*>
<!ELEMENT resource
(version,metadata.resource?,multimedia.reference*,(pseudopartof|resource*|blob))>
<!ATTLIST resource
      %id.attr;
      resource.structure %group.structure;
      physical.location %URI; #IMPLIED
>
<!ELEMENT pseudopartof >
<!ATTLIST pseudopartof
     version.id ID #REQUIRED
     referenced.id ID #REQUIRED
     referenced.version CDATA #IMPLIED
>
<!ELEMENT blob (#PCDATA)>
<!ATTLIST blob
      source.url CDATA #IMPLIED
      mimetype %mimetype; #REQUIRED
      encoding %encoding; #REQUIRED
>
<!ELEMENT multimedia.reference (resource.ref)>
<!ATTLIST multimedia.reference
      substitution.string CDATA #IMPLIED
>


<!-- styles are a big reference to a resource with e.g. style sheets -->
<!-- they can contain sub-styles, meaning: alternative or sequence ofd styles -->
<!ELEMENT styles.list (style)*>
<!ELEMENT style (version,resource.ref?,metadata.resource,sub.style*)>
<!ATTLIST style
```

```
      %id.attr;
      %name.attr;
      group.structure %group.structure;
      %comment.attr;
>

<!ELEMENT sub.style EMPTY >
<!ATTLIST sub.style
      quantity %quantities; "1"
      %reference.attrs;
      %comment.attr;
>



<!-- presentations are hierarchically ordered, each node having a reference
      to a style and ev. references to resources -->
<!ELEMENT presentations.list (presentation)*>
<!ELEMENT presentation
(version,metadata.resource?,style.ref,resource.ref*,presentation*)>
<!ATTLIST presentation
      %id.attr;
      persistence %persistence; 'explicit'
      step CDATA #IMPLIED
      title CDATA #IMPLIED
>



<!-- References: to styles and resources  -->
<!ELEMENT style.ref EMPTY>
<!ATTLIST style.ref
  %reference.attrs;
>
<!ELEMENT resource.ref EMPTY>
<!ATTLIST resource.ref
  %reference.attrs;
>


<!-- version info, included for later versions -->
<!ELEMENT version (version.predesessor?,version.successor*)>
<!ATTLIST version
      version.id CDATA '1.0'
      version.date CDATA #REQUIRED
      version.lockstate %version.lockstate; "unlocked"
      version.creator IDREF #REQUIRED
      version.lockedby CDATA #IMPLIED
      version.lockeddate #IMPLIED
>
<!ELEMENT version.predesessor EMPTY>
<!ATTLIST version.predesessor
  %versionneighbour.attrs;
>
<!ELEMENT version.successor EMPTY>
<!ATTLIST version.successor
  %versionneighbour.attrs;
>
```

Lebenslauf

Name:	Michael Liepert
Geburtstag:	12. März 1966 in Bobingen

Schule und Studium

7/1972 - 5/1985	Gymnasium Bad Waldsee, Deutschland
10/1986 - 8/1993	Dipl.-Inform., Universität Karlsruhe, Deutschland
3/1997 - 12/2001	Dr.-Ing., Technische Universität Darmstadt, Deutschland

Berufliche Tätigkeit

9/1994 - 12/1994	Forschungsinstitut für Informationsverarbeitung und Mustererkennung, Ettlingen; Wissenschaftlicher Angestellter
1/1995 - 12/1996	IBM Deutschland, European Networking Center Heidelberg; Gastwisssenschaftler
1/1997 - 2/1997	IBM Deutschland, Global Services; Entwickler
3/1997 - 1/2002	Technische Universität Darmstadt, KOM (Multimedia Kommunikaton); Wissenschaftlicher Angestellter